A Generative Development Method with Multiple Domain-Specific Languages

Edmilson Campos[1,2], Uirá Kulesza[1], Marília Freire[1,2] and Eduardo Aranha[1]

[1] Federal University of Rio Grande do Norte, Natal-RN, Brazil
{edmilsoncampos,marilia.freire}@ppgsc.ufrn.br
{uira,eduardoaranha}@dimap.ufrn.br

[2] Federal Institute of Rio Grande do Norte, Natal-RN, Brazil.

Abstract. This paper investigates approaches proposed in the literature to compose domain-specific languages (DSLs) and mechanisms to integrate DSLs with feature models (FMs) in product line engineering. We propose a method for the development of generative approaches based on existing related work, which provides guidelines for the systematic development of DSL composition integrated with FMs during domain and application engineering. The proposed method is evaluated through an exploratory study of development of a generative approach for the experimental software engineering domain.

1 INTRODUCTION

Over the last years, there is an increasing usage of domain-specific languages (DSLs) for the development of software systems. The adoption of DSLs raises the abstraction level and provides facilities for the generation of models or source code, especially used in generative approaches to specify and derive products, thus bringing the potential to increase the productivity of software development in various domains [1]. This development requires modeling product families, designing some means to derivative products, i.e. specifying products, providing the implementation components to assemble the products from generators that map the product specifications to concrete assemblies of implementation code assets [2,3]. In this context, DSLs play an important role because they are used to automatically derive products (instances) of a system family.

DSLs can also have different specialization levels. Complex projects require several and different DSLs to specify a complete application. Each DSL can provide a separate view or perspective of the software system modeling. However, the adoption of multiple DSLs brings some consequences that need to be addressed. The main concern is the increased risk of consistency loss among model elements, which requires greater concern with regard to this issue [4]. Consistency maintenance among models is a critical challenge involved on DSLs composition. In this way, new methods, techniques and tools must provide support to these problems.

The use of DSLs in generative development is already considered by some existing methods [5,6]. The generative development aims to specifying, modeling and implementing system families or software product lines so that a given system can be automatically generated from a features specification expressed in some high-level domain-specific language (DSL). Over the past few years, several methods for generative development [6,7,8,5] have been proposed. There is also research work exploring the integration between DSLs and feature models (FMs) [9]. However, most proposed methods for generative development do not explicitly address the issues of DSLs composition. On the other hand, some recent studies [8,9] discuss approaches for software development with multiple DSLs. In this context, this paper proposes a method for the development of generative approaches focused on the integration of FMs with multiple DSLs. The method is based on the investigation of existing research work. The proposed method is evaluated through an exploratory study of implementation of a generative approach for the automatic derivation of workflows for controlled experiments in software engineering.

The remainder of this paper is organized as follow. Section 2 describes our proposed method. Section 3 details the exploratory study of evaluation of the proposed method. Finally, Section 4 presents the conclusions and possible future works.

2 THE PROPOSED METHOD

This section presents our method for generative development, which extends existing methods to deal with DSL composition. Existing research work already considers the usage of DSLs [5] and feature models [9]. However, existing generative development methods do not explicitly address the issues of development with multiple DSLs. Some recent research work [10,11] discusses approaches for development with multiple DSLs. Our proposed method focuses on the integration of FMs and composition of domain-specific languages.

2.1 Method Background

Our method is based on the approach proposed by Czarnecki and Eisenecker [5]. This approach is organized in two main phases from product line engineering (PLE): (i) *domain engineering* (DE) – which focuses on the identification of common and variable features/requirements, the definition of a flexible software product line architecture that addresses the implementation of reusable code assets, and the definition of DSLs that enable the customization of the architecture and code assets for generating specific products; and (ii) *application engineering* (AE) – that includes activities to derive specific products using the DSLs and code assets producing during DE. Our method focuses mainly in the domain design and implementation from DE, and in the product derivation from AE.

Voelter and Visser [7] investigate the application of DSLs in PLE as a middle ground between FM and code assets of the software product line (SPL). They analyze the limits of FM expressiveness and show that DSLs can be used as a complementary element in such cases. DSLs do not represent an alternative mutually exclusive to FMs, they can be used in combination with FMs in order to expand the possibilities of product derivation in SPLs. Since we are interested in investigating different strategies to combine FMs and DSLs in the DE and AE, their approach [7] contributed to the development of our method.

Regarding the adoption of multiple DSLs, several questions still need to be explored and investigated. Hessellund *et al.* [4] conducted a case study to investigate the kinds of existing constraints between DSLs. Four kinds of constraints were identified: (i) *well-formedness of individual artifacts*; (ii) s*imple referential* integrity across artifacts; (iii) *references with additional constraints*; and (iv) *style constraints*. Moreover, constraints violation may be still classified according to the severity level, which can be errors or warnings [12]. Some of these issues are addressed by Hessellund [8], which proposes a method that provides activities for the identification, specification and application of composition between DSLs. This work played an important role in defining the development method of generative approaches proposed in this paper, specifically the activities to compose DSLs.

Fig. 1 presents an overview of our proposed method. The general structure of the method is based on the generative development approach proposed in [5] focusing on the steps that involve the use of DSLs (domain design/implementation and product derivation). There are several activities to identify and implement DSLs from input artifacts such as the domain requirements, FM, and the product line architecture. In addition, specific activities were included at each step to support the development and composition of DSLs as an alternative to the limits of FM expressiveness.

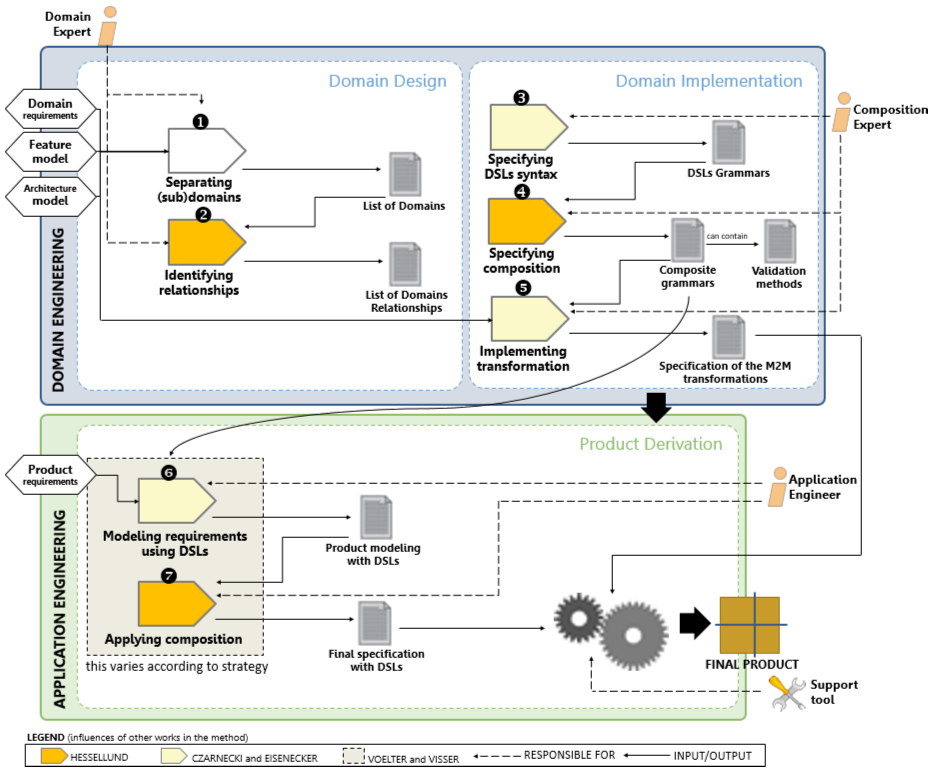


Fig. 1. Method Overview

2.2 Method Structure

The method is divided into three stages: (i) Domain Design, (ii) Domain Implementation; and (iii) Product Derivation. The first two are part of the domain engineering, and the last one of the application engineering. Each stage has activities that are performed by specific roles to produce or generate a set of artifacts. Some activities of our method are based or extend characteristics of other approaches (Section 2.1), which are represented in the legend of Fig. 1. The method also presents alternative strategies to integrate DSLs with FMs: (i) using only DSLs; or (ii) combining DSLs with FMs. The following sections provide an overview of each step.

2.2.1 Domain Design

The aim of this first stage is to identify the existence of different domains and their relationships. It receives as input a FM, the specification of the domain requirements, and an architectural model of the SPL (or system family). The input artifacts are then analyzed by a domain expert: (i) to identify which set of abstractions are more strongly connected to characterize a domain; and (ii) to identify the overlap between them. In summary, the activities of Domain Design are the following:

- *Separation of domains (Activity 1)*: It involves the identification and separation of the SPL scope elements into subdomains based on the input artifacts. The result of this activity will result in a list of domains, each one representing a specific aspect to be separately modeled and (possibly) reused;
- *Identification of relationships (Activity 2)*: After the organization into subdomains, it is necessary to verify the occurrence of references among subdomains and classify these overlaps based on the degree of entanglement of the subdomains. It results in a list of domains and their respective relationships.

This last activity brings consequences to the implementation of the DSLs composition. Once identified, the references must be classified according to the complexity and nature of the intertwining of the models. The classification task is part of the *Identification of relationships* activity and is fundamental to the approach. Based on the kind of reference, engineers can choose how to implement the DSL composition during the domain implementation stage. The references also imply the existence of some kinds of constraints that need to be maintained during the development of applications with multiple DSLs.

2.2.2 Domain Implementation

The goal of this stage is to implement specific solutions for each subdomain identified on previous stage. As a result, it must implement the reusable code assets of the SPL (or system family), as well as the DSLs and respective transformations, which are used to support the automatic product derivation during application engineering. The creation of DSLs, either graphical or textual, arises from the need to broaden the expressiveness of domain abstractions, limited by the representation of the FM [7]. The DSL implementation also involves the choice of technologies to implement its grammar and respective relationships. Moreover, the usage of modularly composable DSL has great advantages over monolithic DSL, including reusability and scalability [5].

The following three activities are accomplished in this stage:

- *Specification of DSLs syntax (Activity 3)*: It is necessary to choose the technology to implement a BNF (Backus-Naur Form) grammar that incorporates the variability of the SPL in each domain. The syntax of the DSL must allow specifying any product to be generated from the SPL reusable assets, including those that are not possible using only FM, but can be addressed using more expressive DSLs;
- *Specification of the composition (Activity 4)*: The purpose of this step is to implement the connections identified previously, according to the reference type. This activity will result in changes to the built grammar and, if necessary, the implementation of additional methods to validate restrictions of the DSLs;
- *Implementation of transformations (Activity 5)*: Finally, it is necessary to implement transformations that allow defining the mapping between the DSLs and the reusable code assets of the SPL. This activity will result in a specification of transformations ready to be executed in AE during the product derivation.

A complementary task to the activity of specification of the composition is the implementation of restrictions. Restrictions must be specified to validate primarily the consistency between the models of the DSLs. In some cases, such restrictions need to be implemented to complement the rules defined by the DSL grammar.

2.2.3 Product Derivation

The product derivation is the final stage of the method and belongs to the application engineering phase. The goal of this step is to generate products (systems) from the artifacts produced in the DE phase. The generation of products may occur in a manual or automated way. Generative development motivates automated product derivation using DSLs. The resultant product – software system – may be a partial or complete final product, according to the derivation strategy used. Our method provides two different strategies to product derivation. The first one adopts only DSLs, and the second one integrates DSLs and FMs. They represent distinct alternatives with different purposes for the same goal.

2.2.3.1 Product Derivation with DSLs

In this strategy, the developed DSLs in domain engineering are used to model products (systems) that we are interested to generate. As we are dealing with a composition approach, each DSL is used to specify a specific part of the final product. In this case, it is also necessary to manage the consistency between the DSLs and validate existing restrictions. Validations are only possible due to the implementation of restrictions tied to DSLs grammars. The modeling of the product using DSLs defines the features that will be part of it. The transformation models implemented in the Domain Implementation stage are used to generate the final product from product modeling using DSLs. The DSLs grammars must address all the variables elements of the SPL. This also allows for model validation before generating the final specification of the product, ensuring that the specified product can be transformed to generate the final product without violates any domain restriction.

2.2.3.2 Product Derivation with FMs and DSLs

This strategy is more appropriate when deriving products with similar characteristics to previous ones already generated in the same approach. In other words, the strategy is used when there is already existing DSL modeling containing fragments that can be reused to generate new and very similar products. Therefore, it is necessary to identify those reusable modeling fragments, and associate them to features in a FM. This initial effort will only occur on the first use of this strategy or during the SPL evolution. The first step in the product derivation for this strategy is to specify a selection of features in the FM. After that, it is generated a partial modeling of the desired product expressed in the DSLs. The application engineer can then edit and complement the specification of its product using the DSLs. Finally, this specification is automatically transformed to generate the code assets corresponding to the request product, similar to the previous strategy.

3 EXPLORATORY STUDY

This section presents an exploratory study conducted aiming to evaluate the proposed method. Our study involves the composition of different DSLs to model controlled experiments in software engineering. Moreover, it has used both product derivation strategies of our method when generating specification of experiments.

3.1 Study Research Questions

The main aim of our exploratory study was to evaluate the proposed method at work, through the development of a generative approach that involves the composition of multiple DSLs. In particular, the exploratory study was developed in order to answer the following research questions:

- **RQ1.** How the composition of DSLs can be specified and implemented during the domain engineering using our method, the xText framework and Ecore models?
- **RQ2.** How to implement the product derivation strategies of our method that involve the DSLs and feature model composition in application engineering?
- **RQ3.** Which kind of reuse can be accomplished with the DSLs in the domain and application engineering?

In order to answer these questions, the proposed method was applied to a real domain to investigate how to compose DSLs during the domain engineering and how to use derivation strategies during the application engineering with DSL composition.

3.2 Target Generative Approach

In our study, we have established the following prerequisites to be addressed by the generative approach to be modeled and implemented: (i) to belong to a real domain documented in the literature; (ii) to have a scope with enough complexity to be sectioned into smaller views, thus allowing reuse of parts and modeling of different DSLs; and (iii) to have requirements and variabilities whose representation the FM was not able to express. We have chosen the domain of controlled experiments in Experimental Software Engineering (ESE). In particular, we apply the proposed method in the development of a model-driven generative approach to generate specialized workflows for each experiment subject according to the experiment design. The approach is under study and development in our research group [13,14,15,16].

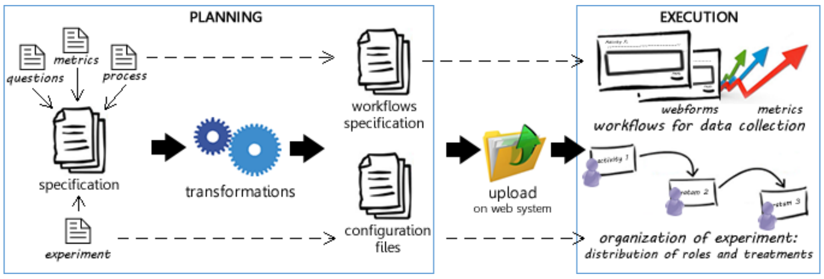


Fig. 2. Controlled experiment approach overview

Fig. 2 presents an overview of the model-driven approach for modeling of controlled experiments. It is presented from two perspectives: planning and execution. The planning perspective aggregates a series of elements used to specify experiments with their respective processes and metrics during the planning phase. This specification is then transformed with the aim of generating workflow specifications and configuration files. After this step, these generated files are deployed in a workflow engine web system in order to generate specialized workflows and web forms, which are responsible for the execution of the experiment and monitoring of the subjects' activities. Fig. 2 also presents this execution perspective. The mapping between the perspectives is implemented by means of model-driven transformations, which are not presented in this paper due to space limitation. In our study, we are more interested in the application of the method to the planning perspective.

3.3 Results Main Summary

This section presents a summary of the main results in terms of produced artifacts during the method application in our exploratory study. More details about these artifacts can be found at http://sites.google.com/site/generativedsl.

Initially, based on the know-how of our research group carrying out controlled experiments [13,15,16,17], we extracted the domain requirements and modeled commonalities and variabilities using a FM. Finally, we also specified the architectural models of the web system responsible for instantiating and executing workflows of the controlled experiment approach [13]. These artifacts – requirements, FM and architecture model – were then provided as input artifacts for our method. The method was applied following its activities in order to produce new artifacts at each stage. A summary of the artifacts produced in each activity is systematically presented below.

Activity 1: *Separating Subdomains*

In this first step, from the feature model and domain requirements specification received as input artifacts, we analyzed the features and grouped them into four subdomains, according to their relationships and reuse opportunity. Fig. 3 shows the FM of our experimental generative approach by highlighting the features from the different subdomains. Each identified subdomain represents a specific view or aspect of the controlled experiment domain that can be reused in other domains. The first one, (i) *Process*, involves the elements that define the procedures to be followed to collect the needed data from the subjects in a controlled experiment. The second, (ii) *Metric*, groups the features that allow specifying metrics related to some of the dependent variables of the specified experiment and that will be collected during its execution. The other, (iii) *Experiment*, is defined for the ESE context and it basically allows setting the treatments and the control variables that are required for the specified experiment. A treatment can be composed of the combination of one or more factors that can have different control levels. In addition, the last one subdomain, (iv) *Questionnaire*, allows specifying questionnaires with the aim of collecting feedback from the experimental subjects.

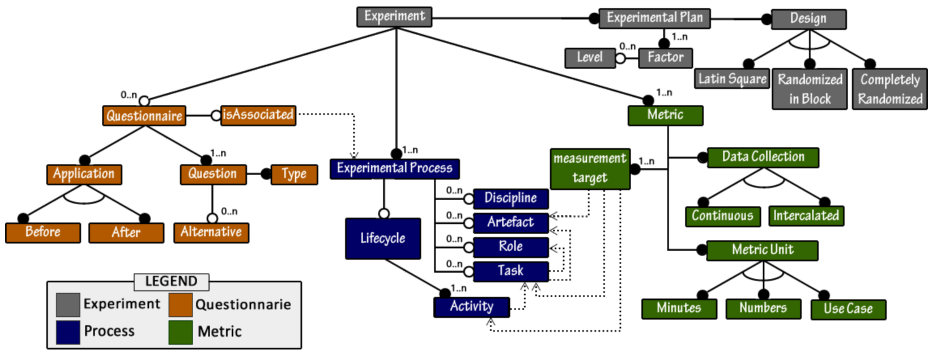


Fig. 3. FM of the experiment approach with subdomains

Activity 2: *Identifying Relationships*

After the separation of the subdomains, it was identified the existing overlaps between them, which were then classified in terms of the degree of complexity of the references. In this activity, the requirement specifications and constraints represented in the FM were used as input artifacts. This analysis resulted in the identification of eight reference points between the subdomains, which are illustrated in Fig. 4 and discussed below.

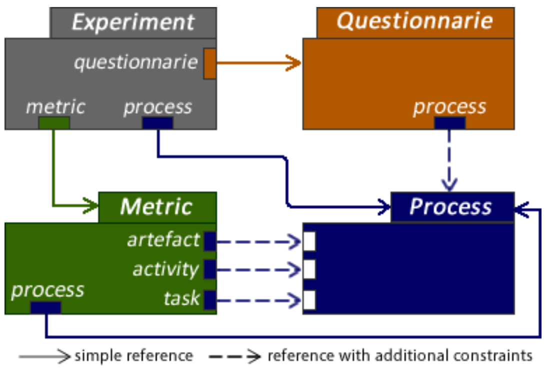


Fig. 4. Relationships between the subdomains

The *Process* subdomain is the only that does not reference another one. For this reason, it can be used to specify processes from distinct domains such as software development or experiment processes, or even a business process. The *Experiment* subdomain needs to reference the processes, metrics and questionnaires that are part of the experiment. A *Questionnaire* element is referenced in an experiment. Each questionnaire may reference one or more processes, because it defines specific questions to be accomplished during the process activities. Moreover, the *Metric* subdomain is always related to a process and must indicate the artifacts, activities or tasks of this process that have to be considered for measurement. Here we identified a typical reference case with additional constraint where the referenced element in the metric has to respect the restriction of being an artifact, activity or task in the existing related process.

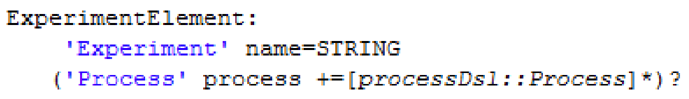
Activity 3: *Specifying DSLs Syntax*

This is the first activity of the implementation stage, which aims to implement the DSLs grammar for each subdomain identified in the previous stage. It also requires selecting the technology to be used. Since one of the goals of this study is to investigate MDE technologies to implement composed DSLs, we chose a model-driven framework for the DSLs development based on Ecore metamodel from the Eclipse Modeling Framework (EMF) [18], known as xText (http://eclipse.org/xtext). Thus, each subdomain resulted in a DSL with its own syntax, which can be used alone or combined.

Activity 4: *Specifying Composition*

After specifying the DSLs grammar, we have specified and codified the references between the DSLs (Activity 2). We changed the DSLs grammar using special features from xText and wrote some additional validations in the Java language. The composition specification varies according to the reference type to be implemented. In our study, two reference types were identified in the investigated domain: (i) simple references and (ii) references with additional constraints.

a) *Simple references*: It was used explicit references, natively supported by xText. The framework offers the possibility that a metamodel imports another one in order to implement the explicit references among models. Since we are working with Ecore-based DSLs, xText has a model generator created for each grammar and responsible for the equivalent Ecore metamodel generation. To perform the importing, it is only needed to inform the grammar model generator path to create the reference. After that, each referenced metamodel can be recognized by an alias name making possible to explicitly refer to anyone of its elements. Fig. 5 illustrates an example of the *ExperimentDSL* grammar referencing the *ProcessDSL process element*.



```
ExperimentElement:
    'Experiment' name=STRING
   ('Process' process +=[processDsl::Process]*)?
```

Fig. 5. *ExperimentDSL* referencing *ProcessDSL*

b) *References with additional constraints*: In this case, we have also used explicit simple references, but it was needed to create extra validation routines. In Ecore-based grammars, the model generator also creates Java classes to each DSL model element beyond helper classes with specific function, such as formatting, validation, and so on. We used this xText support to encode additional restrictions required for a specific DSL. Java methods were implemented to validate this reference type.

Activity 5: *Implementing Transformations*

In this last activity of the domain engineering phase, model-to-model and model-to-text transformations were implemented to transform a ESE controlled experiment specification using the DSLs to workflows and configuration files that represent the experiment to be executed in the workflow engine web system [13]. The QVTo language (http://eclipse.org/mmt) and Acceleo language (http://eclipse.org/acceleo) was used to implement the transformations.

Activity 6: *Modeling the Requirements using DSLs*

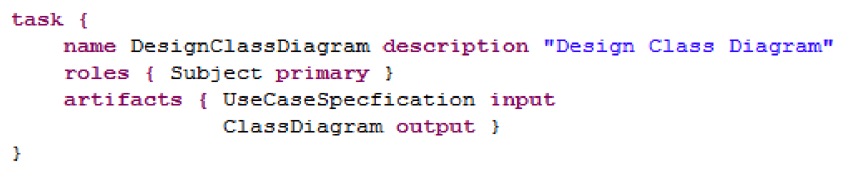
The activity six and seven belong to the application engineering phase. They vary according to the chosen product derivation strategy. Aiming to assess the operation of these activities in both strategies, three controlled experiments were derived twice, one time for each strategy. The experiments were:

Experiment 1: ***Programming Languages***, adapted from [19], the goal is to compare the development productivity using two programming languages.

Experiment 2: ***Configuration Knowledge***, adapted from [20], the goal is to investigate the comprehension of configuration knowledge in three product derivation tools in the SPL.

Experiment 3: ***Testing Techniques for SPL***, adapted from [21], the goal is to analyze the impact of two black-box testing techniques (hereby called treatments) for SPL products derived.

a) *Derivation using only DSLs*: Based on the definition of the scope of each experiment presented, we have used the four DSLs implemented in the domain engineering to model the features to be included in each derived experiment. In this strategy, the modelling was realized manually using the specific resources of the xText framework. Fig. 6 shows a specification example of a task using the *ProcessDSL*.

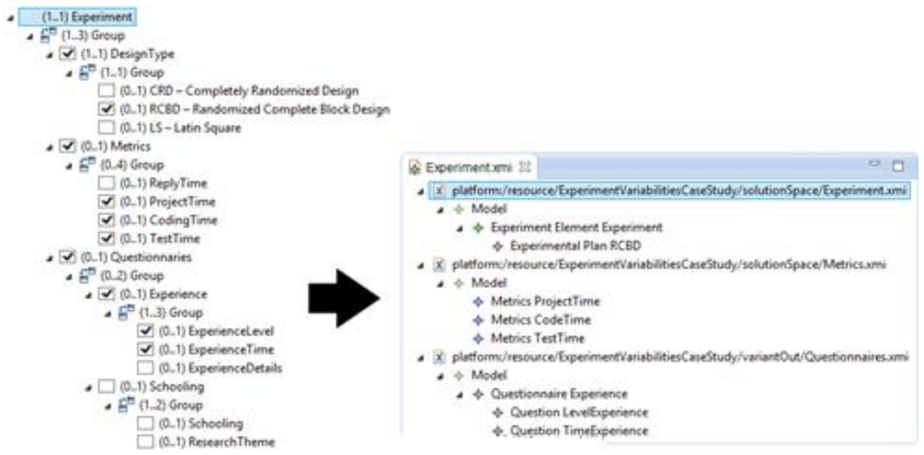


```
task {
    name DesignClassDiagram description "Design Class Diagram"
    roles { Subject primary }
    artifacts { UseCaseSpecfication input
                ClassDiagram output }
}
```

Fig. 6. Specification of a task using the *ProcessDSL*

Each part of the experiment was modeled using the correspondent DSL. The results of these modeling formed the final specification of each experiment to be derived.

b) *Derivation combining DSLs and FMs*: In this strategy, we initially identified reusable modeling fragments and then represented them in a FM. For this, it was used the *FeatureMapper* framework (featuremapper.org) to represent the features in a FM. This FM was used to choose the modeling fragments to be reused and after that a partial modeling of the experiment using the DSL was generated, which can be manually edited to conclude the specification of the experiment. Fig. 7 illustrates an example of this automatic reuse of modeling using the FM built with *FeatureMapper* framework.



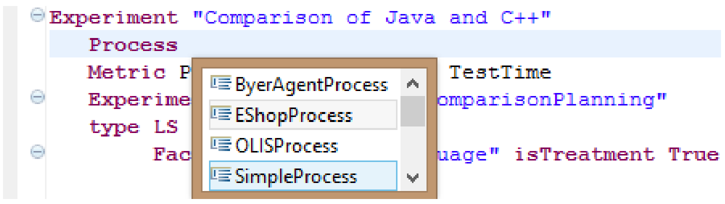
```
(1..1) Experiment
  (1..3) Group
    [✓] (1..1) DesignType
      (1..1) Group
        [ ] (0..1) CRD – Completely Randomized Design
        [✓] (0..1) RCBD – Randomized Complete Block Design
        [ ] (0..1) LS – Latin Square
    [✓] (0..1) Metrics
      (0..4) Group
        [ ] (0..1) ReplyTime
        [✓] (0..1) ProjectTime
        [✓] (0..1) CodingTime
        [✓] (0..1) TestTime
    [✓] (0..1) Questionnaries
      (0..2) Group
        [✓] (0..1) Experience
          (1..3) Group
            [✓] (0..1) ExperienceLevel
            [✓] (0..1) ExperienceTime
            [ ] (0..1) ExperienceDetails
        [ ] (0..1) Schooling
          (1..2) Group
            [ ] (0..1) Schooling
            [ ] (0..1) ResearchTheme
```

```
Experiment.xmi
platform:/resource/ExperimentVariabilitiesCaseStudy/solutionSpace/Experiment.xmi
  Model
    Experiment Element Experiment
      Experimental Plan RCBD
platform:/resource/ExperimentVariabilitiesCaseStudy/solutionSpace/Metrics.xmi
  Model
    Metrics ProjectTime
    Metrics CodeTime
    Metrics TestTime
platform:/resource/ExperimentVariabilitiesCaseStudy/variantOut/Questionnaires.xmi
  Model
    Questionnaire Experience
      Question LevelExperience
      Question TimeExperience
```

Fig. 7. Derivation using *FeatureMapper*

Activity 7: *Applying Composition*

After or during the modeling of the requirements of the experiment to be derived, it is need to compose the models applying the composition between them. This involves the inclusion of resources for (i) navigation between models, (ii) maintenance of the consistency, and (iii) presentation of guidance during the modeling activity. The maintenance checking is the method key-point because it allows examining whether the restrictions are maintained or violated. This consistency checking is consequence of the encoding performed on the specification step, and it becomes visible from the xText alert resources. The xText can provide warnings or errors guidance and also pop-ups with suggested values (Fig. 8) during the typing.

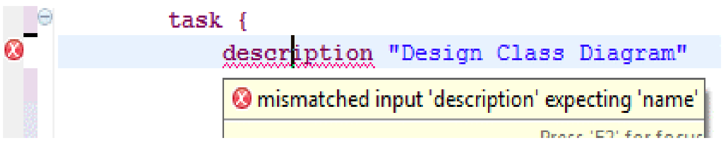


```
Experiment "Comparison of Java and C++"
    Process
    Metric P                          TestTime
    Experime                        omparisonPlanning"
    type LS
        Fac                         uage" isTreatment True
```

```
ByerAgentProcess
EShopProcess
OLISProcess
SimpleProcess
```

Fig. 8. Pop-up with reference suggestions

We also investigated the xText support for the four restrictions types listed in the case study performed in [4]. For the restriction related to the well-formed artefacts issue, the xText uses the DSL grammar to confront the syntax used in the modelling in order to check missing attributes or incorrect syntax. In case of failure, error alerts are displayed, e.g. when a task modelling using *ProcessDSL* does not inform the process name before the description attribute. Fig. 9 shows an example of an error message.



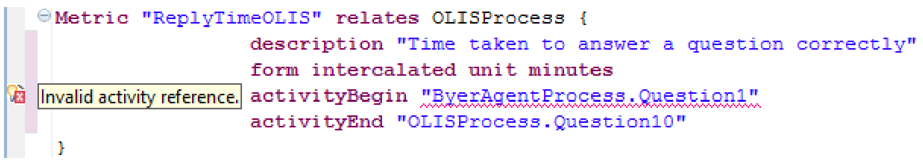
```
task {
    description "Design Class Diagram"
```

```
mismatched input 'description' expecting 'name'
```

Fig. 9. Pop-up indicating not well-defined artefacts

The second restriction type, the simple referential integrity, is the constraint that is better supported by xText. The integrity validation among models only requires code implementation in the case of references with additional restrictions, such as presented in the next paragraph.

References with additional restrictions occurred in our study in the *MetricDSL* and *QuestionnaireDSL*. In this current stage, these restrictions result in similar effects to others, but with customized alerts messages presentation. The warnings and error messages are used to validate the constraints and are introduced through pop-ups conforming encoded in the specification step. Fig. 10 illustrates the metrics modelled using *MetricDSL* for the *Configuration Knowledge Experiment*. We can see that the *ReplyTimeOLIS* metric is related to the *OLISProcess* process but its attribute *activityBegin* refers to an activity (*Question 1*) that belongs to another process – *BuyerAgentProcess*. An error message was displayed as a result of the execution of the validation routine created in the specification step. The error is shown even whether the activity is a valid activity in other process. It does not make sense for this metric to refer to activities from different processes.



```
Metric "ReplyTimeOLIS" relates OLISProcess {
                    description "Time taken to answer a question correctly"
                    form intercalated unit minutes
Invalid activity reference. activityBegin "ByerAgentProcess.Question1"
                    activityEnd "OLISProcess.Question10"
}
```

Fig. 10. Metric modeling violating additional restriction

The last restriction type is the style constraint. The specification of this restriction also requires the implementation of validation methods. Our study has not identified any scenario to apply this kind of restriction. One possible situation to apply it in the experimental generative approach is to create a style that prevents processes to be typed with lowercase start letter. As this rule would not be an actual style, its violation would generate only a warning.

3.4 Discussions

After the usage of the method in our exploratory study, we can reflect about some preliminary results about its applicability. The study investigated the application of mechanisms for the integration of FMs and DSLs using current MDE technologies in order to answer the research questions of this study (Section 3.1).

DSL Composition (RQ1). The results showed that it is possible to apply our method to compose DSLs during domain engineering using technologies based on MDE. In our study, DSLs have been implemented using the xText framework, which is based on the Ecore meta-model. References between grammars were implemented using explicit references between models, which allows the xText to automatically manage the consistency between models during application engineering. Resources native of the xText, such as intelligent assistants, also enabled navigation assistance and models repairs during the modeling, and they were useful to validate the models built with DSLs. Also, the constraints among DSLs were specified and validated by only specifying the DSLs grammar. Just in the case of references with additional constraints and styles constrains that it was necessary to implement extra methods to validate the constraints of the domain using the Java language. Table 1 presents a summary of the implementation that was performed using xText to implement each constraint types.

Table 1. Summary of constraints implementation using xText

Constraint type	Implementation using xText
Well-formedness of individual artifacts	The grammar of the own DSL defines the format of the its element/attribute
Simple referential	Using only explicit references
References with additional constraints	Using explicit references among grammars and extra method to validate the constraint
Style constraints	Using method to validate the style

Derivation Strategies (RQ2). In our study, we derived products using both strategies provided by the method: (i) using only DSLs and (ii) combining DSLs with FMs. As a result, we observed that the first one seems more appropriate to derive new products from a SPL, when there is no reusable modeling fragments between the different products. For instance, when the research teams are still beginning to specify their controlled experiments using the generative approach. On the other hand, the strategy that integrates DSLs with FMs is more appropriate when there are already other similar products derived with model fragments that could be reused. We used the *FeatureMapper* framework to support the implementation of this second strategy. In our study, we have not quantified the spent effort in the application of each strategy. However, we noticed that the strategy combining DSLs and FMs requires additional effort in preparing the FMs. It is important to emphasize that such effort is necessary only in the first application of the strategy, in order to prepare the model fragments to be reused through of their mapping to the FM. Table 2 summarizes the main findings related to the use of each of the strategies in our exploratory study.

Table 2. Comparison of Derivation Strategies

Scenario/Strategy	Only DSLs	DSLs with FMs
Reuse of modeling fragments	Manual (copy-paste)	automatic (selection of features)
Derivation of new products (without fragments to be reused)	most indicated	less indicated
Derivation similar products (with fragments to be reused)	less indicated	most indicated
extra effort needed (in the first application)	none	Specifying the FM to represent variabilities

***Reuse with the DSLs* (RQ3)**. The application of the method also allowed us to observe that the separation of domains into smaller views favors the reuse in the generative approach. In our study, we have applied the method on a specific scenario, but some of our DSLs could be reused in other contexts. The *ExperimentDSL*, for example, can be used to model experiments from other domains. On the other hand, the *MetricDSL* is always related to a process; whereas the *QuestionnaireDSL*, by definition, may or not be related to processes. Finally, the *ProcessDSL* is the only independent DSL in our approach that does not refer to any other. Because of that, it can be reused in different contexts, such as in the modelling of business or software processes. If we consider the reuse inside the same domain – in our case, reuse between ESE controlled experiments – we can notice that the processes, metrics and even questionnaires modeled for a given experiment using our DSLs can be completely or partially reused in the context of other experiments. Hence, despite reuse has not been explicitly investigated in this study, there is a great opportunity to explore the reuse of the specification of an experiment using our DSLs. We are currently conducting new studies to evaluate these issues.

4 CONCLUSIONS AND FUTURE WORK

This paper presented a method for the development of generative approaches with multiple DSLs. Mechanisms to integrate DSLs and FMs, and specifying the DSLs composition were investigated through an exploratory study using the proposed method. Our study focused on the usage of current MDE technologies to provide support to the application of the method, resulting in the composition of Ecore-based DSLs implemented using the xText framework. The method was applied in the modeling and composition of DSLs that allow specifying and executing controlled experiments in software engineering. Our main contributions were: (i) to present a summary of existing approaches to deal with the DSLs composition and generative approaches; (ii) identification and implementation of different integration strategies that can occur between FMs and DSLs during generative development; (iii) proposal of a generative development method that supports multiple DSLs; (iv) evaluation of the proposed method through the design and implementation of a generative approach for the experimental software engineering domain.

We are currently conducting other studies to apply the method to other domains. The main purpose is to evaluate the proposed method considering different settings and contexts. Furthermore, we also intend to conduct more controlled studies with more participants to understand and analyze the method usability, as well as compare quantitatively the different derivation strategies regarding the reuse. Finally, we also plan to evaluate the usage of other MDE frameworks, which can be used to support the development with composition of multiple DSLs.

Acknowledgments. This work was partially supported by National Institute of Science and Technology for Software Engineering (INES) under grants CNPq 573964/2008-4 and 552645/2011-7.

REFERENCES

1. Bräuer, M., Lochmann, H.: Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In: ATEM/MoDELS (2007)
2. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Professional. Addison-Wesley (2011)

3. Weiss, D., Lai, C. T.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional, EUA (1999)
4. Hessellund, A., Czarnecki, K., Wasowski, A.: Guided Development with Multiple Domain-Specific Languages. In : 10th MODELS, Nashville, pp.46-60 (2007)
5. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley Professional, New York, EUA (2000)
6. Greenfield, J., Short, K., Cook, S., *et al.*: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, EUA (2004)
7. Voelter, M., Visser, E.: Product Line Engineering using Domain-Specific Languages. In : 15th SPLC, Washington, pp.70-79 (2011)
8. Hessellund, A.: Domain-specific multimodeling. PhD Thesis, IT University of Copenhagen, Denmark (2009)
9. Lochmann, H., Hessellund, A.: An Integrated View on Modeling with Multiple Domain-Specific Languages. In : IASTED on ICSE, pp.1-10 (2009)
10. Groher, I., Fiege, L., Elsner, C., Schwanninger, C., Völter, M.: Solution-driven software product line engineering. In: Aspect-Oriented Model-Driven Software Product Lines: The AMPLE WAY. Cambridge Univ. Press, NY (2011) 316-344
11. Zschaler, S., Sánchez, P., Nebrera, C., Fuentes, L., Gasiunas, V., Fiege, L.: Produt-driven software product line engineering. In: Aspect-Oriented Model-Driven Software Product Lines: The AMPLE Way. Cambridge University Press, New York, USA (2011) 287-315
12. Bézivin, Jouault, F.: Using ATL for Checking Models. In : GraMoT, 69-81 (2005)
13. Freire, M., Accioly, P., Sizílio, G., Campos Neto, E., Kulesza, U., Aranha, E., Borba, P.: A Model-Driven Approach to Specifying and Monitoring Controlled Experiments in Software Engineering. In : 14th PROFES, Pafos (2013)
14. Freire, M., Aleixo, F., Kulesza, U., Aranha, E., Coelho, R.: Automatic Deployment and Monitoring of Software Processes: A Model-Driven Approach. SEKE, (2011)
15. Campos Neto, E., Freire, M., Kulesza, U., Aranha, E., Bezerra, A.: Composition of Domain Specific Modeling Languages: An Exploratory Study. In: 1st MODELSWARD, Barcelona, vol. 1, pp.149-156 (2013)
16. Freire, M., Kulesza, U., Aranha, E., Jedlitschka, A., Campos Neto, E., *et al.*: An Empirical Study to Evaluate a Domain Specific Language for Formalizing Software Engineering Experiments. In : SEKE, Vancouver, pp.250-255 (2014)
17. Campos Neto, E., Bezerra, A., Freire, M., Kulesza, U., Aranha, E.: Composição de Linguagens de Modelagem Específicas de Domínio: Um Estudo Exploratório. In : III WB-DSDM, Natal, vol. 8, pp.41-48 (2012)
18. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2nd edn. Addison-Wesley Professional (2008)
19. Wohlin, C., dRuneson, P., Höst, M., Ohlsson, M., Wesslén, A.: Experimentation in Software Engineering: An Intoduction. Kluwer Academic Publishers (2000)
20. Cirilo, E., Nunes, I., Garcia, A., Lucena, C.: Configuration Knowledge of Software Product Lines: A Comprehensibility Study. In : VariComp., New York (2011) 1-5
21. Accioly, P., Borba, P., Bonifácio, R.: Comparing Two Black-box Testing Strategies for Software Product Lines. VI SBCARS (2012)