ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 3, No. 6, 358-371 (2018)

www.astesj.com

Special Issue on Recent Advances in Engineering Systems

ASTESJ

ISSN: 2415-6698

A Practical Approach for Extending DSMLs by Composing their Metamodels

Anas Abouzahra*, Ayoub Sabraoui, Karim Afdel

Laboratory of Computer Systems and Vision LabSIV, Ibn Zohr University, Agadir, Morocco

ARTICLE INFO

Article history:
Received: 27 August, 2018
Accepted: 21 November, 2018
Online: 05 December, 2018

Keywords:
Model Driven Engineering
Domain Specific Modeling
Domain Specific Languages
Model Composition
Software Reuse
Code Generation
Experimental Software Engineering

ABSTRACT

Domain specific modeling (DSM) has become popular in the software development field during these last years. It allows to design an application using a domain specific modeling language (DSML) and to generate an end-solution software product directly from models. However providing a new DSML is a complex and costly job. This can be reduced by the reuse of existing DSMLs to compose new ones trough a metamodel composition approach. This paper provides a composition rules based code generator facility for extending DSMLs. In doing so, it proposes three rules to compose DSMLs by composing there metamodels: reference rule, specialization rule and fusion rule. The results of an exploratory case study on using these rules are depicted. In addition a proof of concept of the code generator facility which generates the necessary infrastructure to quickly build new DSMLs is implemented and applied to the case study. The benefits of our approach are measured relying on three indicators: the reduced development time, the reused software components and the gain on learnability.

1. Introduction

This paper is an extension of work originally presented in 2017 European Conference on Electrical Engineering and Computer Science (EECS) [1]. The motivation of this paper is to improve the state of the art of quick development of new DSMLs based on existing ones.

Software composition is a fundamental mean for the evolution of complex software systems [2]. While initial approaches were simply focused on textual composition, more efficient approaches take into account syntax and semantics of the software. There was a tendency over the last twenty years towards operation based composition because of its increased expressiveness. In this direction, Model Driven Engineering (MDE) [3,4] was concerned about improving model composition approaches. From early, the researchers have realized that the application of MDE to complex systems will undoubtedly go through the development of smart and agile model composition techniques [5–9].

A use that takes advantage of model composition is to speed up the implementation of new Domain Specific Modeling Languages (DSML). Designing DSMLS is a not an easy job and generally consuming time [10]. This operation can be simplified by reusing existing DSMLs, composing their metamodels, to get new and larger ones [11]. In fact, the definition of a DSML is based on a metamodel and often provides supporting tools as graphical editors to create and handle models. Therefore, it would be judicious to define the reuse of artifacts at the abstract level; i.e. at metamodels level, then to deduce the projection of this composition at the underneath levels; i.e. the supporting tools.

In the previous work [1], composing metamodels of DSMLs was studied and consequences on their graphical editors were investigated in order to provide a composition of metamodels based approach to extend DSMLs. This work goes further and presents an exploratory study that aims to evaluate the DSML composition approach exposed in [1]. It implements a proof of concept of this approach by developing a code generator facility to make composing graphical editors of DSMLs easier. This prototype provides an automatic code generator which starts from a composition of metamodels of DSMLs, described using composition rules, and generates a layer of code allowing a rapid composition of a new DSML. The gain is measured in terms of development time that can be estimated via the percentage of the generated code. Then, in terms of reused components that can be estimated via the percentage of reused code. As well as in terms of learnability, that can be estimated via the part of kept features and interfaces. These three parameters will be the indicators of evaluation and performance to assess the contribution of this work.

The paper is organized as follows: In Section 2, a series of related works is cited. In Section 3 the problem is stated and the followed methodology is explained. In Section 4 the exploratory

* Anas Abouzahra, Email: abouzahra.anas@gmail.com

study is developed. In Section 5 results are discussed. Finally the Section 6 concludes the paper.

2. Related Works

In this Section a selection of works addressing the composition, extension and reuse of DSMLs in an MDE context is exposed with a brief summary of their features. Moreover, approaches coming from Aspect Oriented Programming (AOP) [12] and Language-oriented programming (LOP) [13] research fields are presented. That is because they have inspired relevant methods for reusing and extending DSMLs. Other approaches that use software product line (SPL) [14] techniques exist [15] but they are minor.

2.1. In MDE

MDE addressed the problem of extending DSMLs by a composition operation. However it varies according to the meaning that each work gives to the composition. This can be a merge operation between models conforming to the same metamodel. As it can be a fusion operation between completely heterogeneous models. It can also be just a resolution of differences between different versions of the same model in order to resolve existing conflicts. Besides, the composition operation can be automatically generated based on mapping calculations or completely custom based on a weaving definition. Furthermore, other approaches provide complementary operations such as checking the consistency of the composition.

Epsilon EML [16] is an Eclipse project which provided a platform for developing substantial and interoperable operations on DSLs among which there is a model composition operation (merging) [17] provided through the Epsilon Merging Language (EML) language [18,19]. EML is applied to compose a number of potentially heterogeneous models. The composition operation is achieved through four steps: comparison, conformance verification, composition and reconciliation. EML was the first language accommodate for model merging and made the case for non-trivial merging of heterogeneous models. However it turned out that it is too verbose for merging homogeneous models. Although, EML is still maintained with significant evolution of its syntax, semantics, capabilities and its underlying platform

AMW [20] is an Eclipse project which proposed a model composition solution (weaving) in parallel of a higher level transformation. In AMW, megamodeling has been introduced to tackle advanced metamodel management, where often the relationships between the metamodels can be considered as composition links. Model weaving has been often used as a solution to compose different DSLs, where the composition is not only the simple gathering of concepts coming from different metamodels, but might also include advanced semantic operators. Unfortunately, and despite all the interest in this tool and its various applications, especially for the traceability of model transformation, the associated eclipse project has been archived. It has not been maintained by the community and no longer by an industrial.

MOMENT [21,22] is a project which aimed to provide a model management platform that furnishes generic operators to handle metamodels described using the Eclipse Modeling Framework (EMF) [23]. In this context, Boronat et al. developed a practical approach for generic model merging. It provides an automate merge operator to merge DSLs artifacts with support for conflict resolution and traceability [24] relying on the QVT Relations language [25]. This work was applied, and specially proven, to class diagrams integration [26].

Melange [27,28] is a project which treated the modularity and the reuse of DSLs and brought a meta-language for implementing DSLs by composing and specializing existing DSL units. It specifies operators for language assembly, for language extensions and language restrictions. Almost introduced operators by are meant to reuse either the semantics or the metamodel as is, in addition of merging code. Except the inheritance operator which is able to modify the initial definitions in the new DSL. Nevertheless, it is not clearly explained how the extended metamodel modify the original one and which concepts can be overridden.

MetaEdit [29,30] is a graphical workbench which provided a language for creating DSMLs. It introduced the concept of joint/linked modeling constructs to reuse DSLs with code generation facility. In MetaEdit+, the code generation is obtained by the use of a template based on the target language. Consequently, it limits the modularity scope of the generation to the modularity capabilities of the target language. Nevertheless, in MetaEdit+ each created DSL is an addition to, or an extension of, the language workbench itself [31]. This extends the capabilities of reuse to DSLs that are already defined in the workbench.

Other works have treated the problem of model composition and reuse from different angles. Indeed Berg et al. in [32] propose an operational semantics based approach for composing and reusing metamodels and models, by including their operational semantics. Composition is performed relaying on a reusable template that permit customizing the metamodel meta-concepts as part of the composition operation. It uses a placeholder mechanism where given meta-concepts of a given metamodel are reused in another metamodel [33]. Schmidt et al. in [34] treated the problem of model composition from a collaborative modeling point of view. They proposed an approach to ease the merging of complex models that are collaboratively developed in teams. This approach aims to furnish collaborative development capabilities in much the same quality as it is provided by version control software or text document merging tools. A recent work in [35] contributed to the same purpose. More, it focused not only on conflicts but also on arbitrary syntactic and semantic consistency issues. Coherent artifacts are merged automatically and only conflicting artifacts are presented to the designer's attention, along with a systematic suggestion of resolution. Otherwise, some works focuses on providing complementary operations to model composition such as checking its consistency. In this direction, Zhang et al. [36] implemented WMCF which is a models composition framework relying on the Alloy language [37,38]. It furnishes a model weaving capability with consistency checking of the resulting composition provided by the Alloy Analyzer.

Besides, generating graphical editors from an abstract definition of a DSL has been addressed by many works. Notably, the *EMF Edit*, the *Graphical Modeling Framework* (GMF) [39,40] and the *Generic Modeling Environment* (GME) [41,42] had brought important contributions. They are mature frameworks based on MDE concepts and furnish tools for defining grammars and generating code for graphical editors.

GMF provides a set of capabilities and runtime infrastructures for generating graphical editors for DSMLs based on their metamodel definitions. Where GME allow decorating a metamodel of a DSML with entities called views. This gathers concepts that will be used to implement models, links between

those concepts and how the concepts will be organized and displayed by the graphical editor. Nevertheless, these frameworks, even if they make the generation of graphical editors of DSLMs much easier, they did not elaborated proper features to support the composition and reuse of DSMLs.

2.2. In AOP

The MDE was much inspired by AOP to deal with the problem of large models for complex systems [43]. The AOP preconizes to design a system by separating the model into different morsels. Each corresponds to a different aspect of the system. This decomposition permits to deal with properties on each aspect before considering the model in its overall. This way we decrease the analysis complexity [44]. However, this requires being able to integrate the morsels of a model with each other's. Thus, AOP has addressed the problem of composition and reuse of models [45].

Hovsepyan et al. [46] elaborated an asymmetric approach to compose artefacts of different DSMLs using an application base model described with UML. This approach was driven by an AOP methodology and was implemented using MDE tools. It introduced the concept of a concern interface which plays the role of a common language between a specific concern and the application base. The composition is then achieved by defining explicitly the syntactic and the semantic relationships between artifacts coming from different concerns.

LARA [47] is a DSL inspired by AOP concepts which brought a novel method for mapping applications to heterogeneous high performance embedded systems. It allows to generate an intermediate aspect representation from a configuration based on different junction points, action models and attributes. This is then given to be processed by the weavers. Pinto et al. [48] has improved LARA by furnishing well-defined library interfaces with concrete implementations for each supported target language. This work contributes to make LARA aspects more concise and improve their reuse. Moreover, it involve to substantial reductions of job effort when developing weavers for new languages.

MATA [49] is an AOP tool built on the top of IBM Rational Software Modeler. It uses model transformation to define and perform composition operations on aspects of a model. The particularity of MATA is that, even if it is inspired by AOP, it did not deal with specific join points. In fact, any model artifact can be considered a join point, and composition is implemented as a special case of model transformation. In addition, critical pair analysis is automatically applied in order to find structural correspondences between various aspects of models. MATA was intended to be a generic approach but it was above all proven on UML models (class diagrams, sequence diagrams and state diagrams) [50].

2.3. In LOP

The LOP field is rich in approaches that ease the design and the reuse of DSLMs. Ones of the most presumably technologies to perform it are the projectional language workbenches. In fact, they provide relevant approaches for extending a DSL and often furnish tools to project it on concrete spaces.

TouchRAM [51] provided a rich client tool for flexible software modelling. It enable at developing reusable and scalable design model through a large registry of design basic design models. It takes advantage from model interfaces and aspect oriented model weaving. The conception of a new design model can be obtained by the composition of available design models in the registry. This work has been improved with TouchCORE [52] which furnish new features for model visualization, model editing model assessment and composition traceability.

MPS [53] provided capabilities to define a DSL trough many aspects: abstract aspects (metamodel), sematic aspects (constraints), concrete syntaxes aspects (graphical editors), generators aspects (model transformations) and many others (e.g. behavior, type system, data flow or intentions) [54]. MPS furnish two ways to reuse DSLs: the reference and the extension mechanisms. The reference consists to use concepts from a given DSL into another one. The extension allows extending a DSL from another one by creating new concepts that inherit all the properties and behavior of their parents [55].

MetaMod [56] is based on a metametamodel that provides metatools to ease the creation and the reuse of DSLs. Convicted that most of simple DSLs do not require more advanced modularity, MetaMod defines the modularity at the value model level provided by the metametamodel itself [57]. Furthermore, having the same modularity mechanisms in many DSLs lead to have robust DSLs, because easier to verify, more fit and easier to learn as well. In addition, this facilitates the reuse of DSLs. However, it limits capabilities of the DSLs if more advanced modularity mechanisms are needed.

Cedalion [58,59] is built on top of Prolog. It provides features for DSLs building with projectional editor trough the description of model aspects such as semantics, structure, projection, and type system definitions from other language workbenches. Cedalion proposes a DSL reuse mechanism. However, because of the close link between the structure of a language and its other aspects, this makes the reuse difficult in Cedalion. In fact, all language aspects of a DSL need to be reused. Nevertheless, extending a DSL with only additional concepts is thus effortless [60].

Spoofax [61] is a language workbench dedicated to design textual DSLs. The platform provides features for code generation, parsers, type checkers, compilers, interpreters, and other tools from language definitions. Spoofax furnishes an API for programmatically composing abstract and concrete syntax of a language. Within Spoofax, the management of modularity can be managed directly in target generated language [62].

Xtext [63] is a textual language workbench based on EMF. It provides tools to define textual DSLs. It furnishes a DSL reuse and extending mechanisms. Reuse permits to cross reference concepts between DSLs. Where, extending allows to a DSL to inherit from another one and to override its grammar rules. However, it allows only to completely overriding them. In addition, this extending mechanism limits a DSL to only extend one other. Regarding the dynamic semantics of the DSL, it can be implemented using other languages such as Xtend [64].

Monticore [65,66] is a textual language workbench. It provides modularity mechanisms that enable the compositional development of textual DSLs and their supporting tools. Especially, inheritance and embedding mechanisms are proposed. Inheritance allow to extend a language where embedding allow to compose different language fragments. Moreover, a special DSL is proposed for the definition of compositional links between languages.

Other works implemented approaches to provide DSMLs composition and reuse capabilities. Pedro et al. [67] have

contributed with an automatic projection approach from metamodels composition patterns into graphical syntax. In [68] they go further more with a definition of operators to compose DSMLs with a proposal for automatic mapping to graphical syntax. Meyers et al. [69,70] proposes a template based technique for the modular definition and composition of DSMLs, including their abstract syntax, semantics, and concrete syntax (relying on metaDepth [71]).

3. Problem Statement & Methodology

Software engineering is essentially involved in providing textual or graphical languages to describe and set out artefacts of a system; their structures, behaviors and interactions. DSMLs provide capabilities to achieve this and allow designers to handle these artefacts as models. Models are intended to be used by tools. Thus, it should be defined a formal description of their concepts. This well-established set of concepts is called a metamodel. This is the principle of DSMLs design. Accordingly, composing DSMLs is primarily a composition of their metamodels.

The composition of metamodels is special issue of a larger problem in MDE, model composition. The composition of the models is a topic of research in continual but very slow evolution in the MDE. This is partly due to the miss of inspiration of patterns from programming languages [72]. It also never has been the subject to standardization like model transformation.

We can define a composition of models simply with a composition operator ⊗ which is a function producing a composed model C by using artifacts of two input models A and B:

$$\otimes : A \times B = C \quad (1)$$

However, model composition can scope various meaning and reach at least three dimensions: abstract syntax, concrete syntax, and semantics [73]. As a DSML is a modeling language we can take inspiration from the composition operation as it is defined in modeling languages; an association of sub languages into one integral language. Where sub artifacts are handled in their original languages and the composed artifact acquire its semantics and its syntax from the composition [74]. In addition, we can draw inspiration from the composition operation as it is defined in programming languages. There is a frank conjunction between the semantic unit (i.e. class) that has a specific interface and the syntactic unit (i.e. file) that is the encapsulation of the implementation [72]. When semantic units are composed, logically de facto the language tool composes the syntactic units. Therefore, a successful approach of composition of DSMLs must deal with the three composition dimensions and maintain the link between abstract syntax (metamodels), concrete syntax and semantics.

In the respect of the aforementioned, this work is an exploratory study whose purpose is to explore means of composing DSMLs by composing their metamodels and studies the projection on their associated graphical editors. Indeed, an appropriate reuse of their syntaxes and graphical editors can be performed. Defining the way that metamodels will be composed implies the way that syntaxes can be merged and editors can be reused. Furthermore, it explores how to automate the composition of graphical editors of the composed DSMLs by implementing a prototype of a composition rules based code generator facility. For that the proposed exploratory study is segmented into five steps as described in Figure 1.

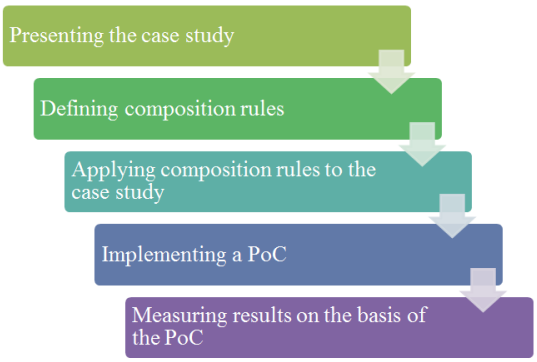


Figure 1. The process of the exploratory study.

A case study is first described. Then the rules for composition of metamodels are defined; on the basis of which DSMLs artifacts can be reused. Later, each is applied to a use case from the case study in order to illustrate it. Next, to prove the concept of this work, an implemented prototype of code generator facility for extending new DSMLs based on the aforementioned composition rules is presented. Finally, three parameters as indicators of evaluation and performance are used to assess the contribution of this study:

- The gain in terms of saved development time that can be estimated via the percentage of the generated code.
- The gain in terms of reused components that can be estimated via the percentage of reused code.
- The gain in terms of learnability, that can be estimated via the part of kept features and interfaces.

These three parameters are measured and discussed in the Section 5.

4. Exploratory Study

4.1. Case Study

Figure 2 represents excerpts of three simple metamodels representing small DSMLs. Each metamodel relies on a different concept.

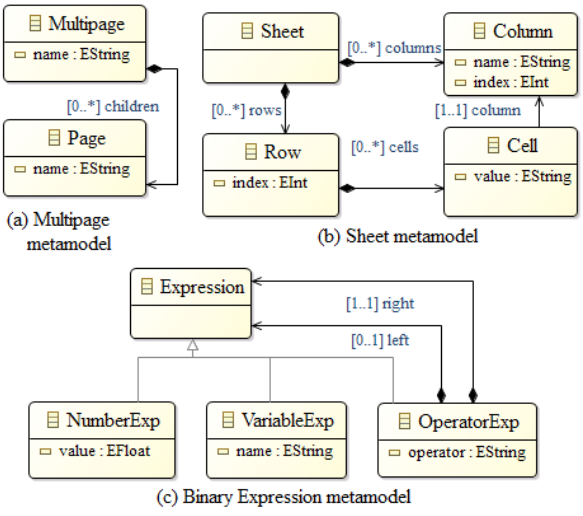
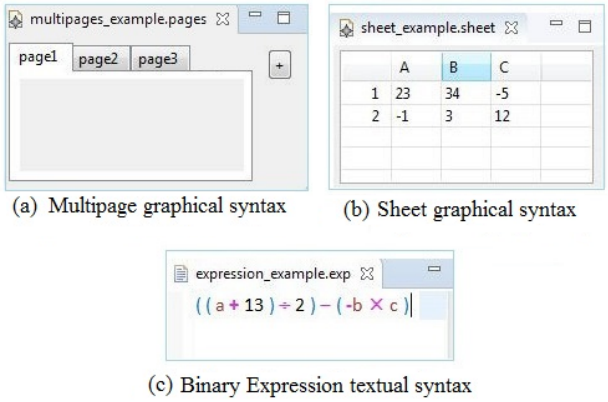


Figure 2. Multipage, Sheet and Expression metamodels.

The first metamodel is the Multipage metamodel ($MM_{Multipage}$). It can be used to describe a multiple page structure. It allows multiple pages to be contained under a single parent page. According to (a), an instance of the metaclass *MultiPage* may contain one or more children instances of the metaclass *Page*. The second metamodel is the Sheet metamodel (MM_{Sheet}). It can be used to describe a sheet containing a two dimension table. According to (b), an instance of the metaclass *Sheet* contains a collection of instances of the metaclass *Row*. Each of which containing a collection of instances of the metaclass *Cell*. In addition, the instance of *Sheet* defines instances of the metaclass *Column*. Each defined instance of *Cell* is related to one of them. The third metamodel is the Binary Expression metamodel (MM_{Exp}). It can be used to describe a binary expression tree which can contain numbers, variables, and unary or binary operators. According to (c), an instance of the metaclass *Expression* is a tree of nodes which can be instances of three metaclasses: *OperatorExp*, *IntegerExp* and *VariableExp*. The *OperatorExp* instances are contained in the internal nodes of the tree, where instances of *IntegerExp* and *VariableExp* are contained in the leaf nodes. Withal, an OperatorExp node may have two children nodes for binary operators (left and right expressions), or one child node for unary operators (only right expression). Each of these DSMLs metamodels relies on a graphical or textual syntax that allows expressing conforming models. Therewith, they are supported by graphical interfaces: editors. Figure 3 shows screen shots of the editors. The graphical syntax of the $MM_{Multipage}$ DSML (a) expresses a *MultiPage* instance as a multiple tabs window. The children instances of *Page* are embedded as a sequence of tabs in the parent *MultiPage* instance. The associated editor has buttons to add new tabs. The graphical syntax of the MM_{Sheet} DSML (b) expresses an instance of *Sheet* as a two dimensions table with indexed instances of *Row* and named instances of *Column*. Instances of *Cell* are represented by the boxes of the table with their *values* inside. The associated editor permits to extend or reduce the table using a hold and move button. This way, the editor allows creating new instances of *Row* and *Column* or deleting existing ones. *Indexes* of *Row* instances are represented at the left side of the table. Editing *names* of instances of *Column* and *values* of instances of *Cell* is done via the textual edition of the related boxes. The textual syntax of the MM_{Exp} DSML (c) expresses an instance of *Expression* using a mathematical grammar where parentheses represent internal nodes. The associated editor is a textual file editor with syntax highlighting.



(a) Multipage graphical syntax (b) Sheet graphical syntax

(c) Binary Expression textual syntax

Figure 3. Multipage, Sheet and Expression concrete syntaxes.

This case study is used later to create step by step a new DSML that meets the following requirements reusing DSMLs (a), (b) and (c):

- *RQ1*. A sheet cell must be able to contain a binary expression.
- *RQ2*. A binary expression defined inside a cell must be able to reference the value of another cell of the sheet's table.
- *RQ3*. A multiple page must be able to be composed of multiple sheet tabs.

In the following Subsections three rules for composing metamodels are defined. Next, they are illustrated relying on the above requirements. Each time a requirement is fulfilled it uses an application of a defined rule. Besides, an investigation of the reuse of the syntax and graphical artifacts of original DSMLs is realized. It aims to obtain an extended DSMLs based on the performed composition of metamodels expressed by means of the proposed composition rules.

4.2. Composition Rules

To describe a composition rule, the following formalism is used :

$$\otimes\text{Rule: } MM_A \times MM_B \ (arguments \ \ldots) = MM_C \qquad (2)$$

Where;

- MM_A and MM_B are metamodels to be composed.
- ⊗Rule is the composition rule.
- MM_C is the composed metamodel.

Reference Rule

A reference rule allows the establishment of discrete connections between instances of a model, conforming to MM_C, defined by concepts coming from MM_A and MM_B. It defines an oriented binary association in MM_C from a metaclass MT_1 of MM_A toward a metaclass MT_2 of MM_B. It is used to connect one instance of MT_1 to many instances of MT_2. It could be a simple link, an aggregation or a composition. It must specify *multiplicity* to mean how many instances of MT_2 can be referenced from an instance of MT_1. The $\otimes Ref$ rule can be defined as follows :

$$\otimes\text{Ref: } MM_A \times MM_B \ (MT_1, MT_2, [m_1, m_2], c_1, c_2) = MM_C \qquad (3)$$

- $[m_1, m2]$ are integers to express multiplicity with a minimum value m_1 and a maximum value m_2.
- c_1 is a Boolean value to mean whether the reference expresses a containment association (i.e. aggregation).
- c_2 is a Boolean value to mean whether the reference expresses a container association (i.e. composition).
- MM_C is the composed metamodel.

Specialization Rule

The specialization rule permits to compose metamodels with an inheritance concept similar to the concept of specialization in object oriented programming. It allows to a metaclass MT_1 from a metamodel MM_A to acquire all the properties and behaviors of another metaclass MT_2 from a metamodel MM_B. Thus, attributes,

associations, or methods can be reused. The ⊗*Spe* rule can be defined as follows :

$$\otimes\text{Spe: } MM_A \times MM_B\,(MT_1, MT_2) = MM_C \quad (4)$$

Fusion Rule

The fusion rule is used to bind metaclasses coming from different metamodels in order to fusion them in the composed metamodel. It allows a metaclass MT_1 from a metamodel MM_A and a metaclass MT_2 from a metamodel MM_B to form a new hybrid metaclass MT_3 in the composed metamodel through over a customized binding. The binding defines the matching between the properties of metaclasses MT_1 and MT_2: attributes references and methods. In addition it specifies those to keep and those to delete. The ⊗*Fus* rule can be defined as follows:

$$\otimes\text{Fus: } MM_A \times MM_B\,(MT_1, MT_2, \{bindings\}) = MM_C \quad (5)$$

4.3. Rules Application

A ⊗*Ref* rule can be applied to fulfill the requirement RQ1. Considering that MT_1 is the *Cell* metaclass as defined in the MM_{Sheet} metamodel and MT_2 is the *Expression* metaclass as defined in the MM_{Exp} metamodel. The ⊗*Ref* rule can be applied as follows:

$$\otimes\text{Ref}_{\text{expression}}: \quad MM_{Sheet} \times MM_{Exp}\,(Cell, Expression, [0, 1], true, true) = MM_{C1} \quad (6)$$

By applying the $\otimes Ref_{expression}$ rule, the composed metamodel MM_{C1} is obtained. MM_{C1} is shown in Figure 4, where the $\otimes Ref_{expression}$ rule is represented with the bold line starting with a lozenge. The designed composition allows an instance of *Cell* to contain an instance of *Expression*. The achieved composition is projected in order to create a new extended graphical editor for the new DSML (d). It is based on the MM_{C1} metamodel and reuses the graphical artifacts of DSMLs (b) and (c). A mock-up of the extended editor of (d) is shown in Figure 4 where the textual editor of the DSML (c) is included in the top of the editor of the DSML (b). According to (d), a sheet's cell is able to contain the value of a binary expression. Therefore, the cell's value is the computed value of an expression which can be edited in the top textual editor.

Similarly, another ⊗*Ref* rule can be applied to fulfill the requirement RQ2. However, this time the ⊗*Ref* rule has to define a simple link reference, from the metaclass *VariableExp* toward the metaclass *Cell*. The ⊗*Ref* rule can be applied as follows :

$$\otimes\text{Ref}_{\text{refersTo}}: \quad MM_{C1} \times MM_{C1}\,(VariableExp, Cell, [0, 1], false, false) = MM_{C2} \quad (7)$$

By applying the $\otimes Ref_{refersTo}$ rule, the composed metamodel MM_{C2} is obtained. MM_{C2} is shown in Figure 4, where the $\otimes Ref_{refersTo}$ rule is represented with the bold arrow. The designed composition allows the definition of cross references between cells expression. In this way, an instance of *Expression* contained inside an instance of *Cell* can reference the value of another instance of *Cell* present in the table. Explained otherwise, an instance of *Expression* which is structured as a tree can include in its nodes a reference to an instance of *Cell* through an instance of *VariableExp*. It is important to observe that this design adapts the *Cell*'s instances to behave as *Expression*'s instances. It is worth mentioning that such pattern, applied with the ⊗*Ref* rule can be useful to solve situations where it is need to adapt concepts of different metamodels when composing them. The achieved composition is projected in order to create a new extended graphical editor for the new DSML (e). It is based on the MM_{C2} metamodel and reuses the graphical artifacts of the DSML (d). A mock-up of the extended editor of (e) is shown in Figure 4. According to (e), a binary expression defined inside a cell must be able to reference the value of another cell of the sheet's table. Therefore, the cell's value is the computed value of an expression which can use the computed values of expressions defined elsewhere in the sheet.

A ⊗*Spe* rule can be applied to fulfill the requirement RQ3. Considering that MT_1 is the *Sheet* metaclass as defined in the MM_{C2} metamodel and MT_2 is the *Page* metaclass as defined in the $MM_{Multipage}$ metamodel. The ⊗*Spe* rule can be applied as follows:

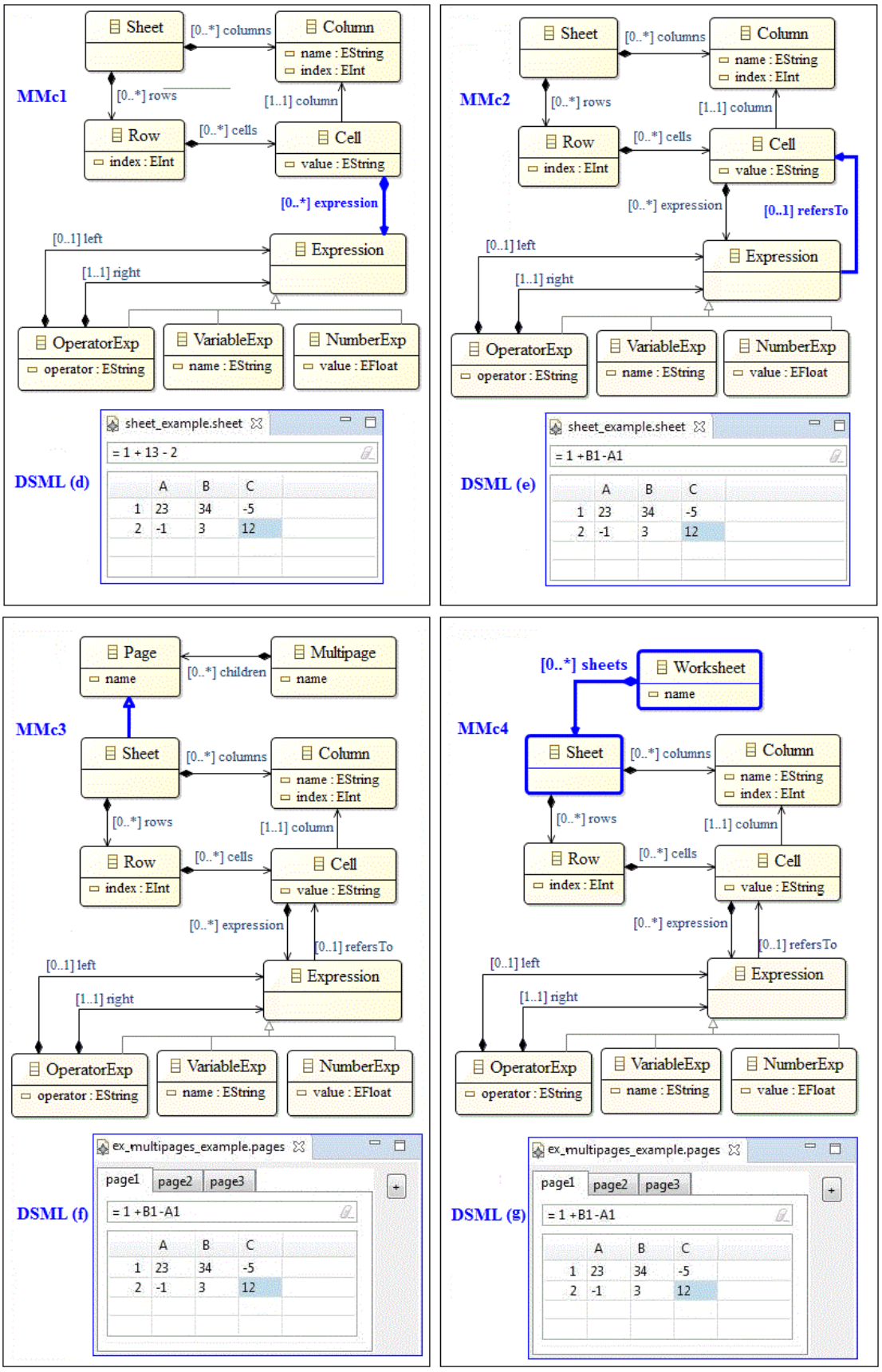
$$\otimes\text{Spe}_{\text{page}}: MMc2 \times MM_{Multipahge}\,(Sheet, Page) = MMc3 \quad (8)$$

By applying the $\otimes Spe_{page}$ rule, the composed metamodel MM_{C3} is obtained. MM_{C3} is shown in Figure 4, where the $\otimes Spe_{page}$ rule is represented with the bold arrow. The designed composition allows a sheet to be a kind of page and then to be a candidate to be a tab of the multiple page. In this way an instance of Multipage can contain instances of Sheet. Therefore, a multiple page is able to be composed of multiple sheet tabs. The achieved composition is projected in order to create a new extended graphical editor for the new DSML (f). It is based on the MM_{C3} metamodel and reuses the graphical artifacts of the DSML (f). A mock-up of the extended editor of (f) is shown in Figure 4. According to (f), it is possible to create multiple tabs of sheets using the means of the multipage editor; i.e. the button that creates pages. The graphical interface of a sheet is then embedded in the graphical container provided for a page and behaves autonomously. However a question may arise about the semantic and utility of the metaclass *Page* in MM_{C3}. The answer depends on the understanding of the requirement RQ3. If it requires obtaining a multiple pages editor composed "only" of sheet tabs. It is probably cleaner to merge metaclasses *Page* and *Sheet*. Moreover, it would be clearer to give a new semantic to the metaclass *Multipage* to indicate that it is a multiple sheets tabs. This leads to define a new rule: the fusion rule.

A ⊗*Fus* rule can fulfill the requirement RQ3 in case it requires obtaining a multiple pages editor composed only of sheet tabs. Considering that MT_1 is the *Sheet* metaclass as defined in the MM_{C3} metamodel and MT_2 is the *Page* metaclass as defined in the $MM_{Multipage}$ metamodel. The ⊗*Fus* rule can be applied as follows:

$$\otimes\text{Fus}_{\text{sheet}}: \quad MMc2 \times MM_{Multipage}(Sheet, Page, \{Sheet.name, Page.name\}) = MMc4 \quad (9)$$

By applying the $\otimes Fus_{sheet}$ rule, the composed metamodel MM_{C4} is obtained. MM_{C4} is shown in Figure 4, where the $\otimes Fus_{sheet}$ rule had the apparent effect to superimpose the metaclasses *Page* and *Sheet* in one metaclass. Additionally, the metaclass *Multipage* can be renamed to *Worksheet* in order to give a better representative name to the container of sheet tabs. The achieved composition projected in order to create the extended graphical editor for the



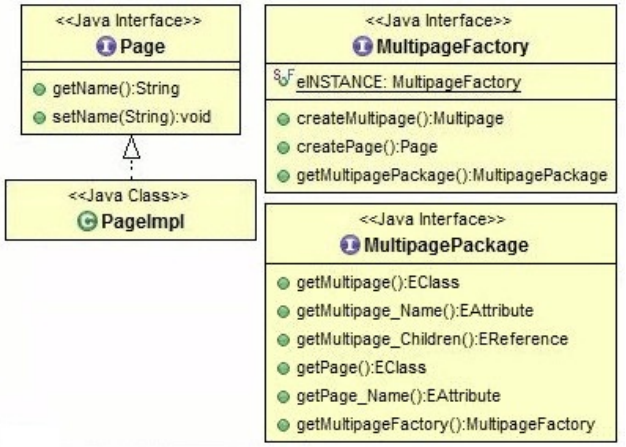
The applications of composition rules on the case study.

new DSML. It is based on the MM_{C4} and leads to the graphical editor of the DSML (g). It very close to the DSML (f) except that the content of the multiple sheets (worksheet) can only be sheet tabs. According to this new DSML, an instance of *Worksheet* can be composed, and only composed, of multiple instances of *Sheet*.

4.4. Proof of Concept

In order to validate the approach exposed in this paper and going further than the theoretical exposition, a Proof of Concept (PoC) was achieved. For this purpose, a prototype of code generator facility based on the aforementioned composition rules was implemented. Then it was applied to our case study. However, before doing so, it is necessary to implement the DSMLs (a) and (b) used in the case study. Next, the metamodels of these DSMLs are composed using one of the rules previously defined. Finally the implemented prototype is used to project the composition onto the graphical editors of DSMLs.

EMF is used to implement the PoC. EMF allows the generation of class architecture that represents metamodel concepts. EMF does not only generate Java classes, but also an associated infrastructure. Thus, one benefits from the persistence of the model in XML Metadata Interchange (XMI) [75] format, but also from a set of tools to handle the model completely independent from the objects it contains. This infrastructure makes it possible to build higher level tools for processing models created with EMF. Within this framework, one of the functionalities is notably the visualization and the edition of models thanks to the EMF Edit framework. Using the capabilities of EMF, the prototype of the PoC aims to generate an overlay layer of code following the EMF code generation. It must provide the necessary infrastructure to make quick building of new DSMLs based on the composition of their metamodels possible.




```
/**
 * Example of generated code usage
 */
// Factory to create instances of model objects
Page page = MultipageFactory.eINSTANCE.createPage();
// Package class to provide access to meta data
EAttribute name_att = MultipagePackage.eINSTANCE
                                      .getPage_Name();
// Set page name using reflective method
page.eSet(name_att, "page 1");
// or direct method
page.setName( "pag 1");
```

Figure 5. A class diagram representing EMF Impl generated classes for the metaclass Page and an example of code usage.

The first step is obviously to implement metamodels. The PoC use case relies only on the two metamodels *Multipage* and *Sheet*. The EMF essential MOF [76] implementation (Ecore) is used to describe metamodels. As aforementioned, EMF provides a code generator facility to generate Java code from a metamodel described in Ecore. It generates, inter-alia, two based packages: One for model implementation (EMF Impl) and another one for graphical user interface editing (EMF Edit). Figure 5 shows an excerpt of the classes generated for the implementation of the *Multipage* metamodel and an example of code usage for creating and manipulating a *Page* instance.

Furthermore, EMF Edit provides capabilities to build a graphical editor. It enables the visualization of model elements and their command-based editing. Figure 6 shows an overview of the architecture of a graphical editor build using the EMF Edit generated code. This will be needed in order to understand the solution exposed further in the paper since it extends the mechanism of EMF Edit. The generated code includes:

- *ItemProvider*(s): They are generated for each class of the metamodel. They are used to display model elements in a graphical editor via an Adapter design pattern (a delegation mechanism that makes it possible to "act as if" an object of type A was an object of type B).
- *ItemProviderAdapterFactory*: It is used to group all *ItemProvider* classes and provide a centralized mechanism to request them.
- *ContentProvider*(s) and *LabelProvider*(s): They are used to provide the display of an item. A *ContentProvider* retrieves the content of an item displayed by a graphical interface where the *LabelProvider* takes care of the visualization (image and text) of the item. The *ContentProvider*(s) and *LabelProvider*(s) can (and usually should) delegate to the same *AdapterFactory* and, therefore, to the same *ItemProvider*(s).
- *ComposedAdapterFactory*: It is useful in order to stick different adapter factories (for individual models).
- *EditingDomain*: It is an editing command structure, including a set of generic command implementation classes to build editors that fully support, cancel and redo actions.

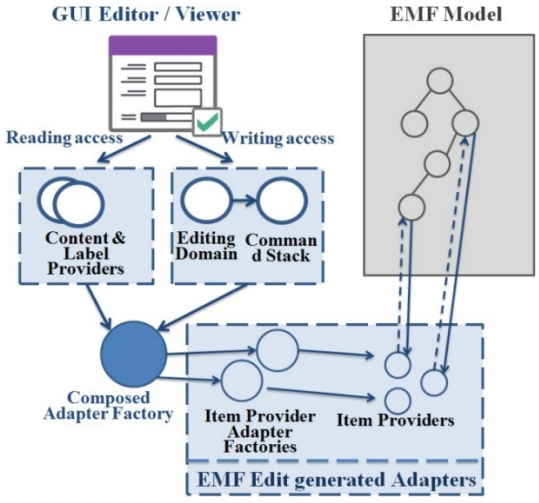


Figure 6. The architecture of a graphical editor built using EMF Edit.

Figure 7 shows a class diagram representing the EMF Edit generated code for the *Page* class from the *Multipage* metamodel.

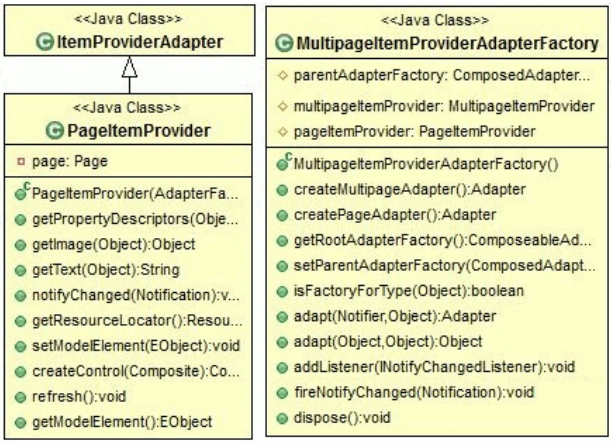


Figure 7. A class diagram representing EMF Edit generated classes for the metaclass Page.

8 shows an excerpt of the *Sheet* editor implementation. It is important to mention that the provision of the graphical content for each element of the model was centralized in a *creatCon-trol()* method attached to its adapter *ItemProvider*. The resulted editors match the screenshots shown in Figure 3.

The code generator facility

As mentioned earlier, the PoC aims to implement a code generator facility that allows generating a Java code overlay of the generated EMF Edit code. It must provide the necessary infrastructure to make it possible to quickly build (compose) new DSMLs based on the composition of their metamodels; using the composition rules defined in Subsection 4.2.

```
/**
 * This is an example of a Sheet model editor.
 */
public class SheetEditor extends EditorPart implements IResourceChangeListener, IResourceDeltaVisitor {
... ...
public void createPartControl(Composite parent) {
// This is the one adapter factory used for providing views of the model.
ComposedAdapterFactor adapterFactory = new ....
SheetItemProvider sItemItemprovider = new SheetItemProvider(adapterFactory);
adapterFactory.addAdapterFactory(sItemItemprovider);
adapterFactory.addAdapterFactory(...);
... ...
// Load the resource through the editing domain.
Sheet sheet = getEditingDomain().getResourceSet().getResource(resourceURI, true).getContents()....;
// Attach the model element to its item provider
sItemItemprovider.setModelElement(sheet);
... ...
// create the Sheet GUI component
adapterFactory.adapt(sheet, SheetItemProvider.class).createControl(parent);
}
... ...
}

/**
 * This is the item provider adapter for a Sheet object.
 * @generated
 */
public class SheetItemProvider extends PageItemProvider {

/** @generated not */
public Composite createControl(Composite parent) {
//Content & Label Provider initialisation
ITreeContentProvider contentProvider =  new SheetViewTreeContentProvider();
DelegatingStyledCellLabelProvider labelProvider = new DelegatingStyledCellLabelProvider(...);
Composite content = new Composite(parent, SWT.NONE);
... ...
// create the model element GUI component
TreeViewer treeViewer = new TreeViewer();
treeViewer.setContentProvider(contentProvider);
treeViewer.setInput((Sheet)getModelElement());

// Iterate over the sheet element to create columns and rows using Content and Label Provides
... ...
return content;
}
... ...
}
```

Figure 8. An excerpt from the code of the Sheet Editor.

Therefore, the adapters mechanism used by EMF Edit is extended with the definition of a Java interface called *IExtendedGraphicalItemProvider*. It specifies a contract of five methods sufficient to create a graphical component connected to an element of the model, to refresh it and to dispose it:

- *createControl()*: It centralizes the provision of the graphical content of the model element.
- *setModelElement()*: It attaches the model element to its provider.
- *getModelElement()*: It accesses the model element from the provider.
- *refresh()*: It refreshes the graphical content of the model element.
- *dispose()*: It disposes the graphical content of the model element.

The implemented code generator facility modifies each edit *ItemAdapter* class, already generated by EMF Edit, in order to make it extend the *IExtendedGraphicalItemProvider* interface. These methods must be implemented and used for the construction of a new graphical editor. So far nothing new compared to the classic use of EMF Edit adapters. Now, if a composition rule is applied between two metamodels, these extended adapters come into play with the use of the aforementioned methods.

Let us consider the following example to better illustrate these statements. Let MM_A and MM_B be two metamodels related to two DSMLs (α) and (β). Let M_A be a model conforming to MM_A and M_B be a model conforming to MM_B. Let $\otimes Rule_C$ be a composition rule MM_A and MM_B in order to create a new DSML (∂). Considering that $\otimes Rule_C$ implies that an element A of the model M_A has to be linked to an element B of the model M_B. A graphical editor built using the architecture shown in Figure 6 makes that the *ItemProviderAdapterFactor* calls the *ItemProvider* IP_A related to the element A to display it in the editor of (α). Likewise, the *ItemProviderAdapterFactor* calls the *ItemProvider* IP_B related to the element B to display it in the editor of (β). In a new architecture built using the extended code generator facility, the generated adapters IP_A and IP_B will be linked with a generated link which reflects the $\otimes Rule_C$ rule. Thus, IP_A will directly call IP_B for displaying element B in the editor of (∂). Figure 9 schematizes this new architecture. For example, a specialization rule will imply, at the generated code, inheritance between IP_A and IP_B. Whereas containment reference rule will imply encapsulation of methods (defined by the interface *IExtendedGraphicalItemProvider*) of IP_B by those of IP_A.

Demonstration of the generator

Let us return back to our case study to illustrate the extended code generator facility through a second example. Let us consider the composition rule $\otimes Spe_{page}$ outlined in Subsection 4.3. The rule was applied on the implemented Ecore metamodels *Sheet* and *Multipage*. Indeed, Ecore makes it possible to describe a link of specialization between two metaclasses of two different metamodels. Then, the EMF generator facility was used to generate the EMF Impl code and the EMF Edit code. Finally, the implemented code generator prototype was used to generate the extension layer with the *ItemProvider*(s) that extend the *IExtendedGraphicalItemProvider* interface. Figure 10 shows a class diagram representing the generated *ItemProvider*(s).

The generated code was used to re-implement the graphical editor of the composed DSML resulting of $\otimes Spe_{page}$. An extended *Multipage* editor has been obtained. It allows a multiple page to be composed of multiple sheet tabs. Figure 11 shows an excerpt from the code of the extended *Multipage* editor. It demonstrates how the *PageItemProvider* delegates the creation of the graphical component of an instance of *Sheet*, included under a *Multipage* instance, to a *SheetItemProvider*. It takes advantage of the polymorphism between the *PageItemProvider* and the *SheetItemProvider* to call the appropriate graphical interface creation method. In the same way, the code of the *MultipageItemProvider* demonstrates how the *refresh* and *dispose* methods can be called for each instance of *Page* contained in an instance of *Multipage*. It takes advantage of polymorphism to apply the appropriate method depending on whether the displayed instance is an instance of *Page* or an instance of *Sheet*.

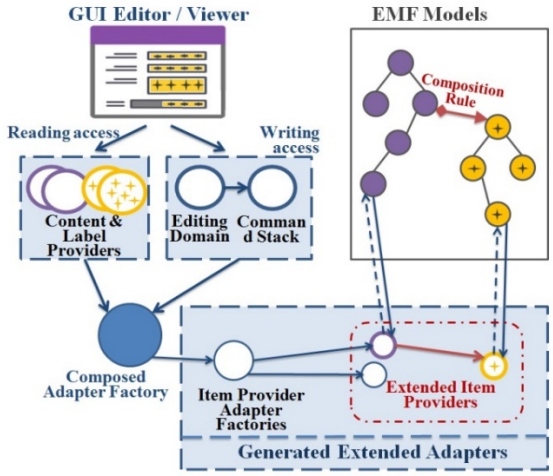


Figure 9. The architecture of a graphical editor built using the extended code generator facility.

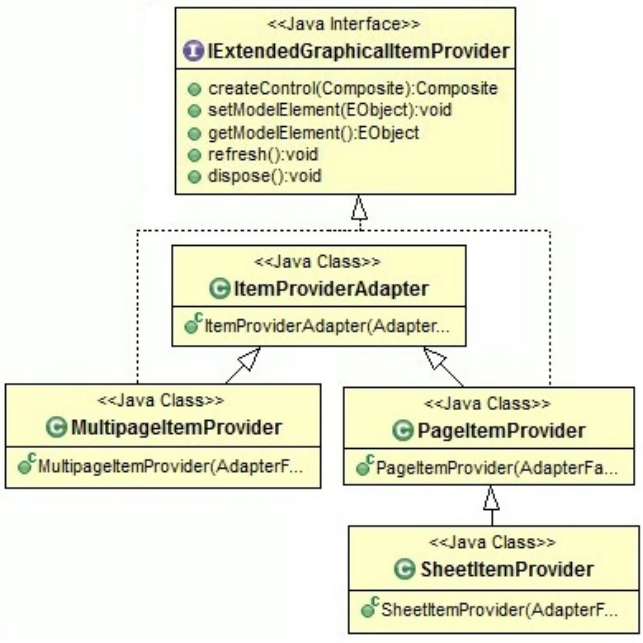


Figure 10. A class diagram representing the generated Item Providers using the extended code generator facility.

In this paper we have conducted an exploratory study whose goal is to explore means of composing DSMLs by composing their

```
/**
 * This is an example of an Extended Multipage model editor.
 */
public class ExtendedMultipageEditor extends EditorPart
                                     implements IResourceChangeListener, IResourceDeltaVisitor {
... ...
public void createPartControl(Composite parent) {
// This is the one adapter factory used for providing views of the model.
ComposedAdapterFactor adapterFactory = new ....
MultipageItemProvider mItemItemprovider = new MultipageItemProvider(adapterFactory);
adapterFactory.addAdapterFactory(mItemItemprovider);
adapterFactory.addAdapterFactory(...);
... ...
// Load the resource through the editing domain.
Multipage multipage = getEditingDomain().getResourceSet()...;
// Attach the model element to its item provider
mItemItemprovider.setModelElement(multipage);
... ...
// create the Multipage GUI component
tabFolder = (TabFolder) mItemItemprovider.createControl(parent);
  for (Page page : multipage.getChildren()) {
    String tabItemName = page.getName();
    TabItem tabItem = new TabItem(tabFolder, SWT.NONE);
    tabItem.setText(tabItemName);
    PageItemProvider pItemItemprovider  = (PageItemProvider) new SheetItemProviderAdapterFactory().
                                          getAdaptedProvider(page, adapterFactory);

    pItemItemprovider.setModelElement(page);

    // The Page Item Provider delegates the creation of the GUI omponent to a Sheet Item Provider
    // in order to create a Sheet GUI component
    Composite tabItemComposite = pItemItemprovider.createControl(tabFolder);
    tabItem.setControl(tabItemComposite);
  }
}
... ...
}
/**
 * This is the item provider adapter for a Multipage object.
 * @generated
 */
public class MultipageItemProvider extends ItemProviderAdapter
                                   implements IItemPropertySource, IItemLabelProvider,
                                              IStructuredItemContentProvider, ITreeItemContentProvider,
                                              IExtendedGraphicalItemProvider, EditingDomainItemProvider{
... ...
/**
 * @generated
 */
@Override
    public void refresh() {
    for (Page page : multipage.getChildren()) {
        // It delegates to the Sheet Item Provider if the page is an instance of Sheet
        adapterFactory.adapt(page, PageItemProvider.class).refresh(parent);
    }
/**
 * @generated
 */
@Override
    public void dispose() {
    for (Page page : multipage.getChildren()) {
        // It delegates to the Sheet Item Provider if the page is an instance of Sheet
        adapterFactory.adapt(page, PageItemProvider.class).dispose(parent);
    }
... ...
}
```

Figure 11. An excerpt from the code of the Extended Multipage Editor.

metamodels and studies the projection on their associated graphical editors. Three rules of composition were defined: Reference, Specification and Fusion. Each has been applied to a case study to illustrate its use. The impact of this composition on graphic editors has been studied. We have shown how the actual syntaxes of the originals DSMLs have been reused and composed to give shape to the concrete syntax of the composed DSML.

5.1. Development Time

Saving time is saving money. Development time is a major factor in software development. Indeed, with the multitude of technology and the vertiginous speed with which languages of programming develop. It is essential to minimize the development time of new software. It is one of the MDE's battle horses as it introduces the necessary abstraction to safeguard knowledge and automate the projection to technological spaces. In our exploratory study we have implemented a code generation prototype that allows taking advantage of our composition rules. It allows generating a layer of code that facilitates the composition of the concrete syntaxes of composed DSMLs. We have implemented it on our case study and we have shown an example in the previous section. After the method usage, we can measure about some preliminary results about the gain obtained by our approach in term of development time by measuring the percentage of generated of code. Because the percentage of code generated is directly correlated to the development time earned. Indeed, the less time spent writing the automatically generated code represents a time gained directly on the development time. In addition, we measure this value on both EMF Edit and our prototype. In this way we show what we also gain compared to the EMF Edit Framework.

Table 1 presents a comparison results summary. The first column lists the global number of line of code of each DSML. The second column compares the number of line of generated code by EMF Edit and by our generator facility. It worth to note that our generator is only used after a composition. Therefore, it has been used only for DSMLs (d), (e), (f) and (g). The third column shows a percentage comparing between the two tools. This last result is exploited in Fig. 12 to show by interpolation the potential gain in terms of development time. It is important to remember that our generator is an overlay of EMF Edit.

5.2. Code Ruse

One of our stated objectives in this study was the reuse of software components. We explored the reuse of existing DSMLs to extend or compose new ones. This reuse is reflected on the code of the obtained DSML. Thus we measured the percentage of reused code each time we extended our DSML in the application of the exploratory study to the case study.

Table 2 shows the percentage of code reuse each time we extended our DSMLs of the case study. It represents what we reused after applying each composition rule.

5.3. Learnability

In our case study, 100% of the graphic components of the original DSMLs were reused. Very few new graphical features have been introduced in composed DSMLs. This is a very important factor for the ease of learning of users. The learnability of software is often overlooked. However, it is the most influential aspect leading to the success of a software application.

In [78], authors noted that experience with similar software is a major dimension of learnability.

Table 1. Comparison between EMF Edit and Our Generator.

	lines of code	*generated Lines*		*% of generated code*	
		EMF Edit	Our Generator	EMF Edit	Our Generator
DSML(a)	1108	803		72%	
DSML(b)	2637	2369		90%	
DSML(c)	2735	1112		41%	
DSML(d)	3581	1532	3481	43%	97%
DSML(e)	3631	1532	3481	42%	96%
DSML(f)	4589	1916	4309	42%	94%
DSML(g)	4535	1901	4264	42%	94%

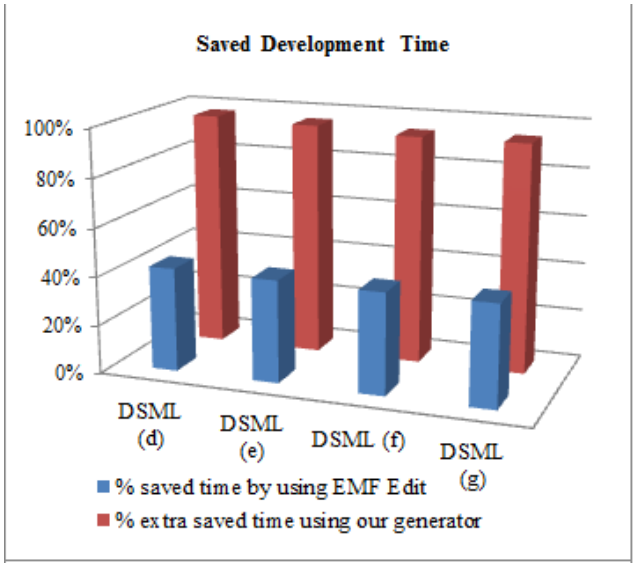


Figure 12. Gain in terms of development time.

Table 2. Percentage of reused after extending DSMLs.

	lines of code	*% of reused code*
DSML(d)	3581	97%
DSML(e)	3631	99%
DSML(f)	4589	97%
DSML(g)	4535	97%

6. Conclusions

This paper investigated the problem of extending DSMLs by composition their metamodels through an exploratory study. It exposes how DSMLs can be reused to rapidly create new ones with low cost. For this purpose three rules to compose DSMLs metamodels were specified: reference, specialization and fusion. A case study was used to illustrate the approach. In addition, the paper presented the implementation of a prototype of a code generator facility based on the aforesaid three composition rules. This prototype is then applied to the case study in order to validate our approach and measure its advantages. Compared to other works, our approach presents advantages, mainly by providing a higher level of reuse of DSMLs artefacts and by providing an automatic generation that facilitates the implementation of DSMLs tools and save development time. In addition, it keeps the graphics interfaces of the original DSMLs thus significantly improving the ease of learning of the new DSMLs.

The main contributions of the paper were: (i) the evaluation of the approach through an exploratory method; and (ii) the

implementation and the experimentation of the code generator facility prototype. Nevertheless, this work is only at its beginning. Indeed, it can be interesting to enlarge the set of composition rules, getting inspired by other principles and patterns coming from modeling languages and programming languages such as: encapsulation, substitution, adaptation and many others. Moreover, it can be interesting to take into account the composability properties of metamodels. Otherwise, the case study used in this study is very simple. It is a choice of writers to better illustrate the approach. However, it can exaggerate the results obtained from the fact of this simplicity.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

We express our respected gratitude goes to Ibn Zohr University, LabSIV laboratory, Ibn Zohr Doctoral Study Center and Faculty of Sciences of Ibn Zohr University for funding this research..

References

[1] A. Abouzahra, A. Sabraoui, K. Afdel, "A Metamodel Composition Driven Approach to Design New Domain Specific Modeling Languages" in 1st European Conference on Electrical Engineering and Computer Science, Bern, Switzerland, 2017. https://doi.org/10.1109/EECS.2017.30

[2] T. Mens, "A State-of-the-Art Survey on Software Merging" IEEE Trans. Softw. Eng. 28(5), 449-462, 2002. https://doi.org/10.1109/TSE.2002.1000449

[3] S. Kent, "Model Driven Engineering" Lect. Notes. Comput. Sc., 2335, 286-298, 2002. https://doi.org/10.1007/3-540-47884-1_16

[4] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering" Computer., 39(2), 25-31,2006.https://doi.org/10.1109/MC.2006.58

[5] R. Reddy, R. France, S. Ghosh, F. Fleurey, B. Baudry, "Providing Support for Model Composition in Metamodels" in 11th IEEE International Enterprise Distributed Object Computing Conference, Annapolis, MD, USA, 2007. https://doi.org/ 10.1109/EDOC.2007.55

[6] J. Estublier, G. Vega, A. D. Ionita, "Composing Domain-Specific Languages for Wide-Scope Software Engineering Applications" Lect. Notes. Comput. Sc., 3713, 69-83, 2005. https://doi.org/10.1007/11557432_6

[7] F. Fleurey, B. Baudry, R. France, S. Ghosh, "A Generic Approach for Automatic Model Composition" Lect. Notes. Comput. Sc., 5002, 7-15, 2008. https://doi.org/10.1007/978-3-540-69073-3_2

[8] J. Bézivin, R. F. Paige, U. Assmann, B. Rumpe, D. C. Schmidt, "Manifesto - Model Engineering for Complex Systems" Dagstuhl Seminar Proceedings, 08331, 2008.

[9] D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. De Lara, I. Ráth, D. Varró, M. Tisi, J. Cabot, "A research roadmap towards achieving scalability in model driven engineering" in Workshop on Scalability in Model Driven Engineering (BigMDE '13), New York, USA, 2013.https://doi.org/10.1145/2487766.2487768

[10] B. Rumpe, "Towards model and language composition" in 1st Workshop on the Globalization of Domain Specific Languages (GlobalDSL '13), New York, USA, 2013.https:// doi.org/10.1145/2489812.2489814

[11] A. Horst, B., Rumpe, "Towards Compositional Domain Specific Languages" Ceur. Workshop. Procee., 1112, 7-16, 2013.

[12] U. Hohenstein and C. Elsner, "Model-driven development versus aspect-oriented programming a case study" in 9th International Conference on Software Paradigm Trends (ICSOFT-PT), Vienna, Austria, 2014. https://doi.org/ 10.5220/0004999901330144

[13] S. Dmitriev, Language oriented programming - the next programming paradigm, http://www.onboard.jetbrains.com/articles/04/10/lop/, accessed 21 November 2018.

[14] J.A. Pereira, K. Constantino, E. Figueiredo, "A Systematic Literature Review of Software Product Line Management Tools" Lect. Notes. Comput. Sc., 8919, 73-89, 2014. https://doi.org/10.1007/978-3-319-14130-5_6

[15] J. White, J. H. Hill, J. Gray, S. Tambe, A. S. Gokhale, D. C. Schmidt, "Improving Domain-Specific Language Reuse with Software Product Line Techniques" IEEE Software., 26(4), 47-53, 2009. https://doi.org/10.1109/MS.2009.95

[16] The Epsilon Homepage, https://www.eclipse.org/epsilon/, accessed 21 November 2018.

[17] The Epsilon Merging Language (EML) Homepage, http://www.eclipse.org/epsilon/doc/eml/, accessed 21 November 2018.

[18] D. S. Kolovos, R. F. Paige, F. A. C. Polack, "Merging models with the epsilon merging language (EML)" Lect. Notes. Comput. Sc., 4199, 215-229, 2006. https://doi.org/10.1007/11880240_16

[19] D. Kolovos, "Merging Models with the Epsilon Merging Language - A Decade Later", in 19th ACM/IEEE International Conference on Model Driven Engineering languages and Systems, Saint-Malo, France, 2016.

[20] M. D. Del Fabro, P. Valduriez, "Towards the efficient development of model transformations using model weaving and matching transformations" Softw. Syst. Model., 8(3), 305-324, 2009. https://doi.org/10.1007/s10270-008-0094-z

[21] The MOMENT web site, http://moment.dsic.upv.es//, accessed 21 November 2018.

[22] A. Boronat, "MOMENT: A Formal Framework for MOdel managemMENT" Ph.D Thesis, Universitat Politècnica de València, 2007.

[23] The Eclipse Modeling Framework (EMF) Homepage, http://www.eclipse.org/modeling/emf/, accessed 21 November 2018.

[24] A. Boronat, J. Á. Carsí, I. Ramos, "Automatic Support for Traceability in a Generic Model Management Framework" Lect. Notes. Comput. Sc., 3748, 316-330, 2005. https://doi.org/10.1007/11581741_23

[25] QVT, The MOF Query/View/Transformation specification page, http://www.omg.org/spec/QVT/, accessed 21 November 2018.

[26] A. Boronat, J. Á. Carsí, I. Ramos, P. Letelier, "Formal model merging applied to class diagram integration" Electron. Notes Theor. Comput. Sci., 166, 5-26, 2007. https://doi.org/10.1016/j.entcs.2006.06.013

[27] T. Degueule, B. Combemale, A. Blouin, O. Barais, J.M. Jézéquel, "Melange: a meta-language for modular and reusable development of DSLs" in 2015 ACM SIGPLAN International Conference on Software Language Engineering (SLE 2015), New York, USA, 2015. https://doi.org/10.1145/2814251.2814252.

[28] T. Degueule, B. Combemale, A. Blouin, O. Barais, J. M. Jézéquel, "Safemodel polymorphism for flexible modeling" Comput. Lang. Syst. Struct, 49(C),176-195, 2017. https://doi.org/10.1016/j.cl.2016.09.001

[29] S. Kelly, K. Lyytinen, M. Rossi, "Metaedit + a fully configurable multi-user and multi-tool case and came environment" Lect. Notes. Comput. Sc., 1080, 1-21, 1996. https://doi.org/10.1007/3-540-61292-0_1

[30] S. Kelly, J. P. Tolvanen, Domain-specific modeling: enabling full code generation , John Wiley & Sons, 2008.

[31] S. Erdweg, T. van der Storm, M. Völter, L. Tratt, R. Bosman, W. R. Cook, A. Gerritsen f, Angelo Hulshout g, Steven Kelly h, Alex Loh c, Gabriël Konat i, Pedro J. Molina j, Martin Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, K. van der Vlist, G. Wachsmuth, J. van der Woning, "Evaluating and comparing language workbenches: existing results and benchmarks for the future" Comput. Lang. Syst. Struct., 44(PA), 24-47, 2015. https://doi.org/10.1016/j.cl.2015.08.007

[32] H. Berg, B. Møller-Pedersen, "Type-Safe Symmetric Composition of Metamodels Using Templates" Lect. Notes. Comput. Sc., 7744, 160-178, 2012. https://doi.org/10.1007/978-3-642-36757-1_10

[33] H. Berg, B. Møller-Pedersen, "Metamodel and Model Composition by Integration of Operational Semantics" Comm. Com. Inf. Sc., 580, 172-189, 2015. https://doi.org/10.1007/978-3-319-27869-8_10

[34] Schmidt, M., Wenzel, S., Kehrer, T., Kelter, U., "History-based merging of models" in 2009 ICSE Workshop on Comparison and Versioning of Software Models (CVSM '09), Washington, USA, 2009. https://doi.org/10.1109/CVSM.2009.5071716

[35] H. K. Dam, A. Egyed, M. Winikoff, A. Reder, R. E. Lopez-Herrejon, "Consistent merging of model versions" J. Syst. Softw., 112(C), 137-155, 2016. https://doi.org/10.1016/j.jss.2015.06.044

[36] D. Zhang, S. Li, X. Liu, "An Approach for Model Composition and Verification" in 1th IEEE Computer Society International Joint Conference on INC, IMS and IDC, Seoul, South Korea, 2009. https://doi.org/10.1109/NCM.2009.271

[37] The Alloy Homepage. http://alloy.mit.edu, accessed 21 November 2018.

[38] D. Jackson, "a lightweight object modelling notation" ACM Trans. Softw. Eng. Methodol., 11(2), 256-290, 2002. https://doi.org/10.1145/505145.505149.

[39] The Eclipse Graphical Modeling Framework (GMF) Homepage, http://www.eclipse.org/modeling/gmf/, accessed 21 November 2018.

[40] The Eclipse Modeling Project (EMP) Homepage, http://www.eclipse.org/modeling/, accessed 21 November 2018.

[41] Á. Lédeczi, A. Bakay, M. Maroti, P. Volgyesi, G. Nordstrom, J. Sprinkle, G. Karsai, "Composing domain-specific design environments", Computer., 34(11), 44-51, 2001. https://doi.org/10.1109/2.963443

[42] Á. Lédeczi, G. Nordstrom, G. Karsai, P. Volgyesi, M. Maroti, "On metamodel composition" in 2001 IEEE International Conference on Control Applications (CCA'01), Mexico City, Mexico, 2001. https://doi.org/10.1109/CCA.2001.973959

[43] J. M. Jézéquel, "Model driven design and aspect weaving" Softw. Syst. Model., 7(2), 209-218, 2008. https://doi.org/10.1007/s10270-008-0080-5
[44] O. Barais, J. Klein, B. Baudry, A. Jackson, S. Clarke, "Composing multi-view aspect models" in 7th International Conference on Composition-Based Software Systems (ICCBSS 2008), Washington, USA, 2008. https://doi.org/10.1109/ICCBSS.2008.12 2008
[45] P. Sánchez, L. Fuentes, D. Stein, S. Hanenberg, R. Unland, "Aspect-oriented model weaving beyond model composition and model transformation" Lect. Notes. Comput. Sc., 5301, 766-781, 2008. https://doi.org/10.1007/978-3-540-87875-9_53
[46] A. Hovsepyan, S. Van Baelen, Y. Berbers, W. Joosen, "Specifying and Composing Concerns Expressed in Domain-Specific Modeling Languages", In M. Oriol, B. Meyer (Ed.), Objects, Components, Models and Patterns, Lect. Notes. Bus. Inf., 33, 116-135, 2009. https://doi.org/10.1007/978-3-642-02571-6_8
[47] M. P. Cardoso, T. Carvalho, J. G. F. Coutinho, W. Luk, R. Nobre, P. Diniz, Z. Petrov, "LARA: an aspect-oriented programming language for embedded systems" in 11th annual international conference on Aspect-oriented Software Development, New York, USA, 2012. https://doi.org/10.1145/2162049.2162071
[48] P. Pinto, T. Carvalho, J. Bispo, M. A. Ramalho, J. M. P. Cardoso, "Aspect composition for multiple target languages using LARA" Comput. Lang. Syst. Struct, 53, 1-26, 2018. https://doi.org/10.1016/j.cl.2017.12.003
[49] J. Whittle, P. Jayaraman, A. Elkhodary, A. Moreira, J. Araúj, "MATA: A unified approach for composing UML aspect models based on graph transformation" Lect. Notes. Comput. Sc., 5560, 191-237, 2009. https://doi.org/10.1007/978-3-642-03764-1_6
[50] J. Whittle, P. Jayaraman, "MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation" Lect. Notes. Comput. Sc., 5002, 16-27, 2008. https://doi.org/10.1007/978-3-540-69073-3_3
[51] M. Schöttle, O. Alam, FP. Garcia, G. Mussbacher, J. Kienzle, "TouchRAM: a multitouch-enabled software design tool supporting concern-oriented reuse" in 13th International Conference on Modularity. New York, USA, 2014. https://doi.org/10.1145/2584469.2584475
[52] M. Schöttle, N. Thimmegowda, O. Alam, J. Kienzle, G. Mussbacher, "Feature modelling and traceability for concern-driven software development with TouchCORE" in 14th International Conference on Modularity, New York, USA, 2015. https:// doi.org/10.1145/2735386.2735922
[53] M. Voelter, "Language and IDE modularization, extension and composition with MPS" Lect. Notes. Comput. Sc., 7680, 383-430, 2013. https://doi.org/10.1007/978-3-642-35992-7_11
[54] M. Voelter, J. Warmer, B. Kolb, "Projecting a modular future" IEEE. Software., 32(5), 46-52, 2015. https://doi.org/10.1109/MS.2014.103
[55] M. Voelter, B. Kolb, T. Szabó, D. Ratiu, A. van Deursen, "Lessons learned from developing mbeddr: a case study in language engineering with MPS" Softw. Syst. Model., 17(66), 1-46, 2017. https://doi.org/10.1007/s10270-016-0575-4
[56] A. M. Şutîi, "MetaMod: a modeling formalism with modularity at its core" in 30th IEEE/ACM International Conference on Automated Software Engineering, Lincoln, USA, 2015. https://doi.org/10.1109/ASE.2015.29
[57] A. M. Şutîi, M. Van Den Brand, T. Verhoeff, "Exploration of modularity and reusability of domain-specific languages: an expression DSL in MetaMod" Comput. Lang. Syst. Struct., 51(C), 48-70, 2018. https://doi.org/10.1016/j.cl.2017.07.004
[58] D. H. Lorenz, B. Rosenan, "Cedalion: a language for language oriented programming" SIGPLAN. Not., 46(10), 733-752, 2011. https://doi.org/10.1145/2076021.2048123
[59] D. H. Lorenz, B. Rosenan, "Code reuse with language oriented programming" Lect. Notes. Comput. Sc., 6727, 167-182, 2011. https://doi.org/10.1007/978-3-642-21347-2_13
[60] D. H., Lorenz, B. Rosenan, "CEDALIONs Response to the 2016 Language Workbench Challenge", in LWC@SLE 2016 Language Workbench Challenge at the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, Amsterdam, Netherlands, 2016.
[61] L. C. L. Kats, E. Visser, "The Spoofax language workbench: rules for declarative specification of languages and IDEs" SIGPLAN. Not., 45(10), 444-463, 2010. https://doi.org/10.1145/1932682.1869497
[62] M. Voelter, S. Benz, C. Dietrich, B. Engelmann, M. Helander, L. C. L. Kats Kats, E. Visser, G. Wachsmuth, DSL engineering: Designing, implementing and using domain-specific languages, dslbook.org, 2013.
[63] The Xtext Homepage, http://www.eclipse.org/Xtext/, accessed 21 November 2018.
[64] L. Bettini, Implementing domain-specific languages with Xtext and Xtend, Packt Publishing Ltd, 2016.
[65] H. Krahn, B. Rumpe, S. Völkel, "Monticore: Modular development of textual domain specific languages" Lect. Notes. Bus. Inf., 11, 297-315, 2008. https://doi.org/10.1007/978-3-540-69824-1_17
[66] H. Krahn, B. Rumpe, S. Völkel, "MontiCore: a framework for compositional development of domain specific languages" Int. J. Softw. Tools Technol. Transf., 12(5), 353-372, 2010. https://doi.org/10.1007/s10009-010-0142-1
[67] L. Pedro, V. Amaral, D. Buchs, "Foundations for a domain specific modeling language prototyping environment: A compositional approach" in 8th OOPSLA workshop on domain-specific modeling. In Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications (OOPSLA Companion '08). ACM, New York, USA, 2008. https://doi.org/10.1145/1449814.1449886
[68] L. Pedro, M. Risoldi, D. Buchs, B. Barroca, V. Amaral, "Composing Visual Syntax for Domain Specific Languages" Lect. Notes. Comput. Sc., 5611, 889-898, 2009. https://doi.org/10.1007/978-3-642-02577-8_97
[69] B. Meyers, "A Multi-Paradigm Modelling Approach for the Engineering of Modelling Languages" Ceur. Workshop. Procee., 1321, 2-9, 2015.
[70] B. Meyers, A. Cicchetti, E. Guerra, J. de Lara, "Composing textual modelling languages in practice" in 6th International Workshop on Multi-Paradigm Modeling, New York, USA, 2012. https://doi.org/10.1145/2508443.2508449
[71] J. De Lara, E. Guerra, "Deep metamodelling with metaDepth" Lect. Notes. Comput. Sc., 6141, 1-20, 2009. https://doi.org/10.1007/978-3-642-13953-6_1
[72] C. Herrmann, H. Krahn, B, Rumpe, M. Schindler, S. Völkel, "An Algebraic View on the Semantics of Model Composition" Lect. Notes. Comput. Sc., 4530, 99-113, 2007. https://doi.org/10.1007/978-3-540-72901-3_8
[73] S. Kelly, J. P. Tolvanen, Domain-Specific Modeling: Enabling Full Code Generation, Wiley, 2008.
[74] S. Völkel, "Kompositionale entwicklung domänenspezifischer sprachen," Ph.D Thesis, Technical University Carolo-Wilhelmina, 2011.
[75] The Xml Metadata Interchange (XMI) Specification page, http://www.omg.org/mof/ http://www.omg.org/spec/XMI/, accessed 21 November 2018.
[76] The MetaObject Facility (MOF) Specification page, http://www.omg.org/mof/, accessed 21 November 2018.
[77] The Eclipse Standard Widget Toolkit (SWT) Homepage, https://www.eclipse.org/swt/, accessed 21 November 2018.
[78] T. Grossman, G. W. Fitzmaurice, R. Attar, "A survey of software learnability: metrics, methodologies and guidelines" in 27th International Conference on Human Factors in Computing Systems, New York, USA, 2009. https://doi.org/10.1145/1518701.1518803