Task Completeness Assessments in the Evolution of Domain-Specific Modelling Languages

Vijanti Ramautar(✉), Sergio España, and Sjaak Brinkkemper

Department of Information and Computing Sciences, Utrecht University, Princetonplein 5, Utrecht 3584 CC, The Netherlands
{v.d.ramautar,s.espana,s.brinkkemper}@uu.nl

Abstract. [**Background**] Domain-specific modelling languages (DSMLs) are tailored to particular application domains and are common in model-driven information system engineering. To support new modelling requirements, increase the maturity of the languages, and keep them relevant to their domain, DSMLs need to be evolved. [**Aims**] Since little is known regarding the complexity of the evolution process, in this paper, we investigate which incompletions are prevalent in each DSML evolution activity. [**Method**] We conduct a quantitative empirical study where the object of study, a DSML in the domain of ethical, social and environmental accounting, is supported by a metamodel in UML and a textual grammar in Xtext. Ninety-two participants grouped in 25 teams have evolved the DSML based on a set of new requirements, updating the metamodel and the grammar. We assess the completeness of each evolution activity and identify incompletions per artefact. We have also enquired the participants about their perceptions of the evolution process. [**Results**] The completeness of the metamodel evolution activity is about 1.25 times higher than it is for the grammar. The metamodelling primitives that are more likely to cause problems are relationships and enumerations. With respect to the Xtext grammars most incompletions are localised in rule calls, cross references and cardinalities. This is consistent with the participants' perceptions about the difficulty of each activity and primitive. [**Contribution**] Our findings are relevant for the design and testing of DSMLs, as well as for education on DSMLs.

Keywords: Model-driven information systems engineering · domain-specific modelling language · evolution · Xtext grammar · metamodel

1 Introduction

Model-driven information systems engineering (MDE) aims at decreasing the developer effort, reducing time-to-market, and reducing development complexity [14,37]. In some project situations, creating a domain-specific modelling language (DSML) is a convenient approach [25]; for instance, when engineering a software product line where a family of related information systems (ISs) is developed, or when user-modelling is intended. DSMLs are one family of domain-specific languages (DSL), where the goal leans towards modelling rather than programming. DSLs can be designed in many application domains, and in contrast to general-purpose languages, DSLs provide the right level of abstraction and expression for the problem domain, while providing critical domain properties [20]. DSML and DSL engineering methodology has garnered much attention from practitioners and academia and several development methods exist (e.g. in [5]). DSMLs are expressed by several types of formalisms, where metamodels and textual grammars are two common artefacts, the former often specifying the abstract syntax and the latter often specifying the concrete syntax of the DSML [13]. Visual notations are another approach to concrete syntaxes [22]. In such project situations, a DSML evolution will require interrelated activities to update the metamodel and the grammar or visual notation. In this paper, we focus on metamodels and grammars.

While DSL evolution and maintenance have been the focus of several investigations [38], this practice has received less attention in the case of DSMLs. Moreover, we miss insights on the degradation of DSMLs due to evolution activities. The introduction of incompletions in the DSML artefacts is likely to affect the process of the underlying IS engineering endeavours, as well as the quality of the resulting system. Understanding, anticipating, preventing, and resolving incompletions in the DSML artefacts that arise as a result of an evolution process becomes paramount for the overall success of IS engineering practices involving DSMLs.

This paper aims to investigate which incompletions are prevalent in each DSML evolution activity. We conduct an empirical study where teams of participants are requested to evolve a given DSML based on a set of new requirements, and then asked about their perceptions of the process. Following the traditional metrics of task completeness in usability studies [36], we define (task) incompletion as the lack of an element that should have been added, updated or deleted to implement the requirement.

The main contributions are: (i) an assessment of the prevalence of incompletions in each DSML evolution activity (and its corresponding artefact), (ii) an analysis of how the incompletions are located in the primitives of the metamodel and the grammar, (iii) an analysis of the perceptions of the participants, also in relation to their performance. We analyse whether there is a relationship between problems that arise during metamodel evolution and issues that arise during grammar evolution. The scientific contributions can be employed by DSML engineers and teachers to identify where to put emphasis (e.g. elaborate explanations, training, and documentation) during DSML development and evolution. Moreover, the results can provide the focus for DSML testing to discover whether DSML evolution is likely to be problematic. Lastly, DSML engineers and researchers can use our results to prevent complications during model-driven information systems engineering projects.

The paper is structured as follows. Section 2 presents background information. The empirical study design is explained in Sect. 3. We present and discuss the results of the empirical study in Sect. 4. In Sect. 5 we address threats to validity and formulate our conclusions.

2 Background

2.1 Related Work

One of the most common MDE approaches entails transforming a model, using model transformation technologies to derive a software product [30]. The models that provide the input for MDE are created with a DSL. Mernik et al. [25] define four types of DSLs: i) DSLs with well-defined execution semantics, (ii) input languages of an application generator, (iii) DSLs not primarily meant to be executable but nevertheless useful for application generation, and (iv) DSLs not meant to be executable, for instance, domain-specific data structure representations. Examples of approaches for DSL implementation are using an interpreter or a compiler. In this article, we focus on a DSL of the fourth type (referred to as DSML), which is later interpreted in runtime.

Typical artefacts in DSML engineering are metamodels for specifying the abstract syntax of the DSML [13], and textual grammars for specifying its concrete syntax. Frank emphasises the importance of selecting a suitable metamodelling language [11]. Moreover, he describes DSML development as a major effort that requires a clear division of labour. The tasks can be divided over roles including domain expert, user, business analyst, language designer, tool expert, graphic artist, etc [11]. Herrmannsdoerfer et al. researched motivations behind language evolution and found that metamodels evolve due to user requests and technological changes [15]. Furthermore, metamodel changes are very likely to impact artefacts which are directly related to them (e.g. models and transformators). They conclude that language evolution is similar to software evolution.

We found that in the literary body DSL evolution is often understood as the co-evolution of parsers, textual syntax and graphical syntax editors, compilers, code generators, etc. In this article, we focus on metamodel and textual grammar evolution. DSL studies mainly focus on the domain analysis, design and implementation phases, conclude Kosar et al. [19]. They found that DSL maintenance/evolution is insufficiently investigated. Another systematic mapping study on DSL evolution, specifically, concludes that DSL evolution is a topic of increasing relevancy [39].

Scientific literature on approaches for DSL evolutions states that the usability of a DSL is crucial for its maintainability and adaptability [1], discusses the challenge of integrating several existing DSLs into one single and provides tools for automated DSL integration [7,18,24,26,28], explores the possibility of improving DSL evolution through composition [4,34], proposes community-driven language development [16], and weighs the costs and benefits of developing a DSL [6]. The typical activities known from software maintenance also apply to metamodel maintenance [15,23]. Other related work presents high-level insights gained by analysing a case of DSML evolution [2]. Our work contributes to the body of knowledge by pinpointing more exactly where challenges in DSL evolution occur.

2.2 The Application Domain of Ethical, Social Environmental Accounting

The approach studied in this article entails that the models are interpreted by an interpreter. The interpreter recognises constructs in the model and interprets these using an instruction cycle [25]. Our empirical study uses a DSML in the application domain of ethical, social and environmental accounting (ESEA). ESEA is the process of assessing and reporting on an organisation's ethical, social and environmental (ESE) performance [12]. ESEA methods provide guidelines on how to perform the accounting. The accounting results are typically published in an annual sustainability (or non-financial, or integrated) report.

The models created with our DSML contain information on the ESEA method (e.g. the name of the method, the topics assessed, and the indicators used to measure the ESE performance). We refer to this DSML as the openESEA DSML. The models produced with the DSML can be uploaded to an open-source interpreter, also called openESEA[1] [33]. The interpreter parses the models and reacts by activating the appropriate features and displaying the contents of the model. The openESEA DSML consists of a metamodel, which is a UML Class Diagram [29] and a textual Xtext grammar [3]. We use a metamodel as part of our DSML, given that metamodelling is a commonly used approach to capture the abstract syntax of a domain [10,17]. We follow [8] to conceptualise the domain, and express the concrete syntax with a textual grammar. In our approach, we always first evolve the metamodel and document the changes, only then we evolve the textual grammar based on the changes in the metamodel. We do not prescribe this specific approach for evolving DSMLs. However, most DSMLs have at least a metamodel or textual grammar. Hence our results are insightful in the contexts of DSMLs that are specified exclusively with a metamodel or grammar, and for DSMLs specified with both.

3 Quantitative Empirical Study Design

3.1 Research Questions and Objective

The aim of our research is to find out which problems arise during DSML evolution, hence we formulate the following research questions.
MRQ: Which incompletions are prevalent in DSML evolution activities?

RQ1.1: What is the task completion of DSML evolution activities?
RQ1.2: Which primitives of the metamodel and grammar specification languages are more likely to result in incompletion?

[1] https://github.com/sergioespana/openESEA.

We run a quantitative empirical study [41] to identify incompletions in DSML evolution. The objective, expressed with the Goal/Question/Metric method [40], is to (i) analyse incompletions that arise during DSML evolution activities, and (ii) to discover how the process of DSML evolution can be improved. The purpose of our study is to identify which aspects of DSML evolution require emphasis in DSML education and DSML testing. We do this by measuring the completeness of the models constructed, from the point of view of DSML engineers, in the context of extending the openESEA DSML.

3.2 DSML Evolution Process

The DSML evolution process, shown in Fig. 1 is based on earlier research in DSML development and evolution [8,9,33]. This process comprises six activities. During the first activity, new requirements are elicited (A1), previously implemented and backlogged requirements can provide input for this activity. The output is a collection of new requirements that will provide the backbone for the DSML evolution. Next, in A2, the metamodel is evolved, resulting in a new version of the metamodel. The corresponding documentation, explaining each element of the DSML, is updated (A3) to reflect the changes. In A4, the Xtext grammar is evolved, so it corresponds to the new version of the metamodel. Then, the interpreter is updated (A5), to ensure that it can parse and interpret the models that comply with the new grammar version. Lastly, the model is updated (A6) so that it adheres to the grammar. We investigate incompletions in activities A2, A3, A4, and A6. Strictly speaking, activities A2, A3, and A4 are performed by DSML engineers, whereas activity A6 is performed by the DSML users (i.e. IS engineers, domain modellers, etc.).

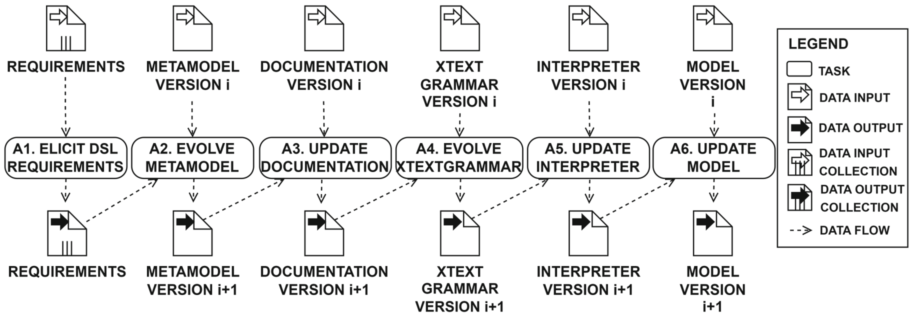


Fig. 1. The DSML evolution process

3.3 Object of Study: openESEA DSML

Every DSML engineering project can have a different infrastructure, but we consider an infrastructure such as in Fig. 2. This figure is a metamodel of DSMLs and details the deliverables involved in the DSML evolution process as seen in Fig. 1.

For simplification purposes, we left out the documentation, models, and interpreter. The **object of study** is the openESEA DSML, which is specified with a metamodel, and an Xtext grammar. All changes to the DSML are documented in an openESEA DSML manual. New requirements lead to a metamodel and grammar change. In case of a metamodel change, a metamodel element has to be changed. Metamodel elements are common primitives, used in UML class diagrams [29]. In case of a grammar change, the changes affect grammar elements, which can either be rules or elements of rules. Our Xtext grammar comprises standard Xtext prmitives [3].

The users of the openESEA technology use the DSML to specify ESEA methods which will later be supported by the interpreter. The metamodel is shown in Fig. 3, and the grammar can be found in a technical report [32]. `ESEA methods` typically consist of a set of `Topics` (e.g. gender diversity and greenhouse gas emission), the topics are measured using `Direct indicators` (e.g. number of non-binary employees) and `Indirect indicators` (e.g. non-binary to male ratio). Data is collected using `Surveys`, which consist of `Sections` and `Questions`. In case of a multiple choice question, `Answer options` can be defined. Some methods issue certification, if the certification requirements are met. Multiple `Certification levels` can be obtained (e.g. bronze, silver, or gold certification). To automatically validate the data, `Validation rules` can be defined. A validation rule can, for instance, state that the number of women in executive positions cannot be greater than the number of women staff.

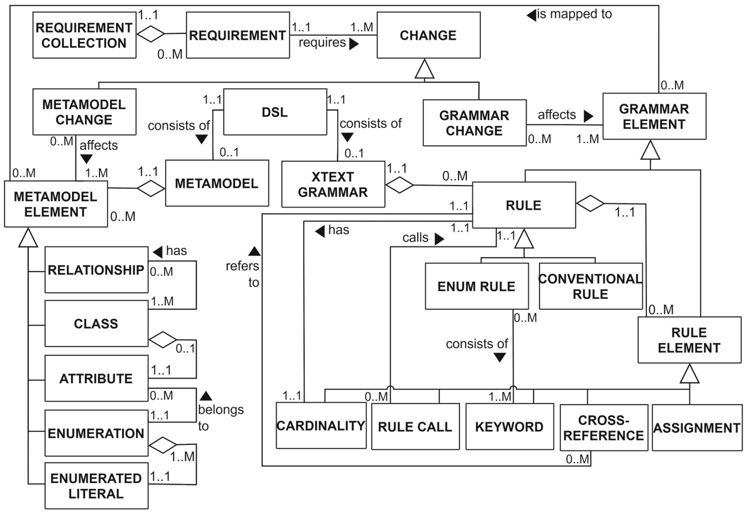


Fig. 2. Metamodel of the deliverables involved in the DSML evolution process

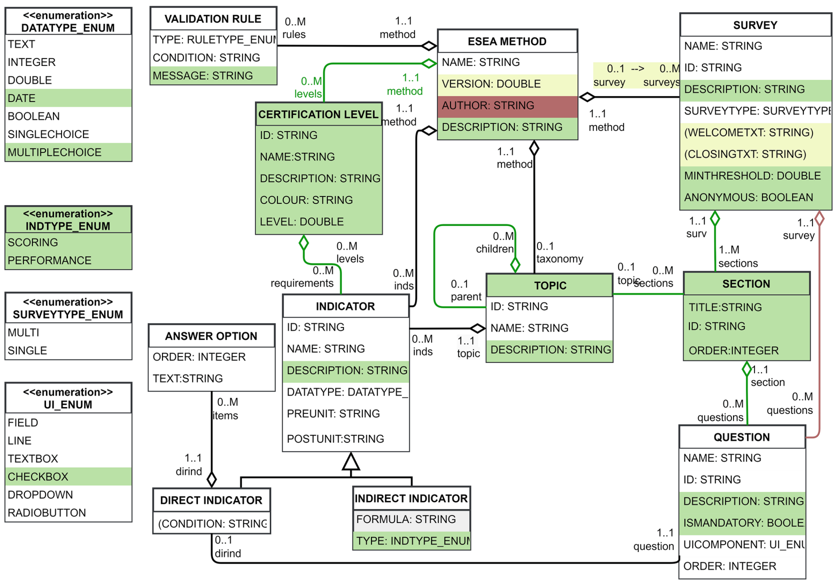


Fig. 3. The openESEA metamodel, with colour-coded changes required of study participants: green depicts additions, yellow updates and red deletions (Color figure online)

3.4 Subjects

We selected subjects using convenience sampling [41]. The subjects are students of the Business Informatics master's programme from Utrecht University (the Netherlands). The empirical study is performed as part of a course on Responsible ICT and is repeated over two years (2021 and 2022). In 2021 45 students, divided over 12 teams, participated; in 2022, 47 students divided over 13 teams participated. The students are educated in method engineering, but prior to the study, they had no knowledge of DSML development and ESEA.

3.5 Empirical Study Protocol

In the empirical study, the participants are asked to perform activities A2, A3, A4, and A6 of the DSML evolution process (see Fig. 1). Before they start the evolution process, they receive training, during which they are familiarised with DSML development, including metamodelling and Xtext grammar development. After the training, they receive an initial metamodel, initial documentation and 16 requirements. Two examples of requirements can be found in Table 1. Based on the requirements, they have to evolve the metamodel and update the documentation accordingly. The requirements can be implemented by adding, updating or removing attributes, relationships, cardinalities, and classes. If all requirements are implemented correctly the metamodel should resemble the reference model in Fig. 3. Next, the participants evolve the Xtext grammar based on the

Table 1. Two examples of requirements (full list in the technical report [32])

ID	Requirement
R5	Right now, the topics are just a flat list. There is also the need to convert that into a taxonomy; that is, a tree of topics. This way, the information is more structured
R14	Based on conversations with a network that could become a future client, we have discovered that their method has indicators whose datatype is a date. We would like to support this in the next version, so we have better chances of convincing them to use our tool

Table 2. The number of metamodel and grammar elements that are necessary to implement each of the requirements.

	Requirement ID																
	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16	**Total**
Metamodel element																	37
Class	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	2
Attribute	1	5	1	1	-	-	1	1	2	-	1	1	-	-	1	4	19
Relationship	-	-	-	-	1 / 2	1	-	-	3	1	-	-	-	-	-	2	8 / 9
Enumeration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	1
Enumerated literal	-	-	-	-	-	-	-	-	-	-		-	1	3	2	-	6
Grammar element																	34
Cross-reference	-	-	-	-	1	-	-	-	-	1	-	-	-	-	-	1	3
Keyword	1	5	1	1	-	-	1	1	2	-	1	-	1	-	1 / 0	4	19 / 18
Rule call	-	-	-	-	0 / 1	1	-	-	3	-	-	-	-	2	-	1	7 / 8
Enum rule	-	-	-	-	-	-	-	-	-	-	-	-	-	1	0 / 1	-	1 / 2
Assignment	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	1	2
Cardinality	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	1

same set of requirements. They receive the reference solution of the metamodel and documentation in advance. This prevents them from making mistakes in their Xtext grammar evolution due to a mistake in their metamodel and documentation solutions. Table 2 shows how each of the requirements can be implemented in the metamodel and grammar. For instance, implementing requirement 5 (R5) requires the participant to add a relationship in the metamodel, and one cross-reference in the grammar. R15 and R5 can be implemented in multiple ways. In R5, another relationship can be updated and a rule call can be added. R15 requires either adding a keyword or an enum rule in the grammar. To assess artefacts created by the participants, we have created a template where, for each requirement, we have listed all the implementations we had conceived beforehand. This acted as a completeness benchmark. We then systematically checked whether the participant had implemented each requirement and; if so, which solution they had opted for, and whether they implemented it correctly.

While we were prepared to extend our list of possible implementations in case they had come up with one that we had not conceived, this did not happen.

After updating the Xtext DSML, the participants should update a model of an ESEA method to make it compliant with the new version of the DSML and showcase the changes they made in the grammar. The DSML evolution activities are performed in groups of four participants, with a couple of teams having fewer participants to accommodate course-related exceptional circumstances. After the DSML evolution tasks, the participants are asked to fill in an extended version of the Method Evaluation Model (MEM) questionnaire [27] where they reflect on the evolution process.

3.6 Variables

For each participant team, we inspect the four evolved DSML artefacts (i.e. metamodel, documentation, Xtext grammar, and model) to identify incompletions. That is, elements that should have been added, updated or deleted to implement the requirements, but the team failed to. This allows us to measure their *completeness* by dividing the number of correctly implemented changes per primitive or artefact by the number of required changes per primitive or artefact.

Apart from completeness, we measured the *perceived ease of use*, *perceived usefulness*, and *intention to use* by deploying a 5-point Likert scale based on the MEM questionnaire. Moreover, we asked the participants to indicate their perceived difficulty of the metamodel and grammar elements, and to reflect on the evolution process in an open question.

3.7 Analysis Procedure

We analyse the task completeness per metamodel and grammar primitive. Moreover, we know the mapping between elements of the metamodel and elements of the grammar; this relationship is labelled "is mapped to" in Fig. 2 and quantified in Table 2. This allows us to also analyse which metamodel primitives suffer from more incompletions when implemented in the grammar. The MEM questionnaire responses are quantitatively analysed and illustrated in box plots. Regarding the open question, we used Nvivo [31] to analyse the responses by creating a taxonomy of codes (e.g. a code for each metamodel primitive) and mapping the part of the response that discusses the code.

4 Results

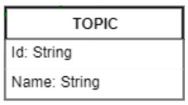
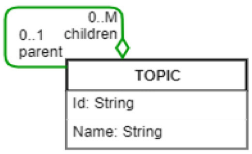
4.1 Metamodel and Documentation Completeness

The completeness of the evolution activities of the metamodel and documentation is shown in Table 3. The completeness is the lowest for the *enumeration* and the *relationship* elements. Requirements related to the enumeration element entailed creating new enumerations, including the enumerated literals. The

Table 3. The completeness percentages for the metamodel (Meta) and documentation (Docu), aggregated per metamodel primitive (Meta primitive)

		2021 (n=12)		2022 (n=13)	
		Meta	**Docu**	**Meta**	**Docu**
Meta primitive	Class	100.0%	98.1%	100.0%	100%
	Attribute	92.6%	93.6%	95.5%	98.3%
	Enumeration literal	82.3%	73.1%	87.7%	72.4%
	Enumeration	61.5%	69.2%	84.6%	46.2%
	Relationship	64.5%	74.6%	70.2%	77.5%

Table 4. Left, the initial metamodel and grammar fragments for R5, and right their solutions.

Initial metamodel/grammar	Solution metamodel/grammar
	 0..M children 0..1 parent TOPIC Id: String Name: String

`Topic:`
`    'topic_id:' name=ID`
`    'Name:' STRING`
`;`
	`Topic:`
	`    'topic_id:' name=ID`
	`    'Name:' STRING`
	`    ('ParentTopic:'linkParentTopic=[Topic])?;`

relationship-related tasks entailed adding or deleting relationships and updating cardinalities. As an example, implementing requirement 5 entailed adding a reflective relationship to the `Topic` class to support a taxonomy of topics. The initial metaclass and metaclass in the reference solution are shown in Table 4. Evidently, these types of evolution activities resulted in the most incompletions. The overall completeness for the metamodel and documentation evolution, shown in Fig. 4, is quite high (>80%). Strikingly, in some cases, the completeness is higher for the documentation evolution than for the metamodel evolution. This indicates that the participants were able to document the required change well, but they did not implement them correctly in the metamodel. Strictly speaking, when following the sequence of activities as described in Fig. 1 this should not be possible.

4.2 Grammar and Model Completeness

The task completeness of the grammar and model yields Table 5. The left table shows the grammar and model task completeness per grammar element, whereas the right depicts the grammar task completeness per metamodel element. We express the grammar completeness also in terms of metamodel elements because metamodel elements are commonly known and intuitive. This analysis shows that implementing relationships in Xtext caused most incompletions, which is in line with our findings presented in Sect. 4.1. When examining the table on the left, the overall completeness of the metamodel solutions is 1.25 times higher than that of the grammar solutions, and Fig. 4 shows that the completeness of the grammar evolution activity is the lowest of all artefacts. *Cross-references*, *cardinalities*, and *rule calls* are the source of most incompletions. To provide an example of what a cross-reference entails, we present the example in Table 4, which shows the initial and solution grammar fragments related to requirement 5. The requirement can be implemented in Xtext by adding a cross-reference, which links a parent topic to a subtopic, using the parent topic ID. Such cross-references have proven to be difficult for the participants. The other element that causes incompletions is the rule call, which is a rule that calls on another rule. For implementing requirement 14, the participants had to create a new rule named `MultipleChoice`, comprising of two keywords and one rule call to the rule `AnswerOption`. Moreover, they had to create a call to the rule `MultipleChoice` in the `Datatype` rule. Such tasks resulted in low completeness. The other element with low overall task completeness is *cardinality*. Rule calls, cardinalities, and cross-references generally refer to relationships in the metamodel. Given that participants had problems modelling relationships in the metamodel, it seems rather logical that these primitives resulted in incompletions. The participants also updated a model to reflect the grammar changes. In some cases, the model task completeness is higher than the grammar completeness. In practice, this is an invalid outcome since the model has to adhere to the grammar rules. However, we noticed that participants divided tasks among the members of their group, where some group members only evolved the grammar, whereas others only updated the model, resulting in models that do not adhere to the team's grammar.

Table 5. Left, the task completeness of the grammar per grammar primitive. Right, the task completeness of the grammar per metamodel primitive.

		2021 (n=12)		**2022 (n=13)**	
		Gram.	**Model**	**Gram.**	**Model**
G. primitive	Keyword	93.6%	84.6%	94.0%	80.1%
	Assignment	82.1%	87.5%	100.0%	82.1%
	Enum rule	87.5%	33.3%	98.2%	35.7%
	Cardinality	39.3%	80.6%	65.2%	73.2%
	Rule call	43.1%	29.2%	40.0%	59.6%
	Cross-reference	22.3%	43.8%	39.3%	50.0%

		2021 n= 12	**2022 n=13**
		Gram.	**Gram.**
M. primitive	Class	82.1%	100.0%
	Attribute	97.4%	98.9%
	Enumeration literal	98.2%	100,0%
	Enumeration	75.0%	96.4%
	Relationship	43.4%	51.1%

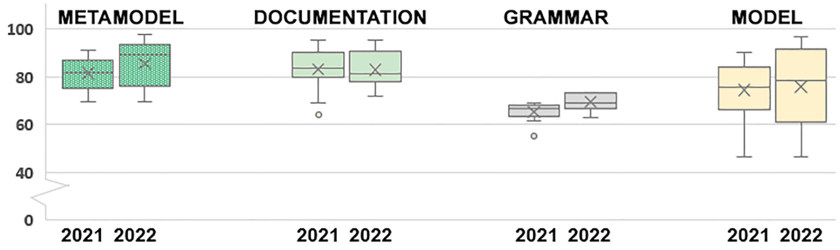


Fig. 4. The completeness per artefact

4.3 Participant Perceptions

The responses to the extended MEM questionnaire reveal the participants' perceptions about three aspects of DSML evolution.

Aspect 1: Difficulty Per Element. In general, the participants perceived evolving the metamodel as quite easy. They described the task as "doable", "intuitive", and "not too difficult". They found updating the documentation rather dull and repetitive, but they understood the necessity for it and acknowledged that it helped them understand the requirements better.

The majority of the students perceived evolving the Xtext grammar as the most difficult step in the evolution process. They found the Xtext framework difficult to work with and the rules were hard to grasp. They described the Xtext grammar as somewhat abstract, and expressed that they were having trouble staying consistent in their application of the rules. Moreover they expressed that they were often able to pinpoint which lines had to be edited, but it was hard to figure out how the lines had to be edited.

In the questionnaire we asked the students to indicate how difficult they found each of the metamodel and grammar primitives, the results are presented in Fig. 5. The perceived difficulty seems to align with the task completeness. The students indicated that they found *relationships* in metamodels and *cardinalities*, *rule calls*, and *cross-references* in grammars the most difficult. The task completeness is the lowest for these elements. There seems to be a correlation between the perceived difficulty of elements and the task completeness of those elements. However, more data points are needed to prove such a relationship.

Aspect 2: Sequence of Activities. The participants were positive about the sequence of evolution activities. They stressed that the metamodel was essential for evolving the Xtext grammar. A quote by a student that emphasises this reads: *"While reading the metamodel I figured [out how] some parts of the grammar should look and operate. Without the metamodel, the grammar alone would be dramatic to understand. The documentation helped in understanding the metamodel itself since [it] showed what the classes in the metamodel did and what attributes they consisted of".*

Aspect 3: Perceived Efficacy and Intention to Use. The results of the MEM questionnaire are shown in Fig. 6. Five is the highest score, indicating

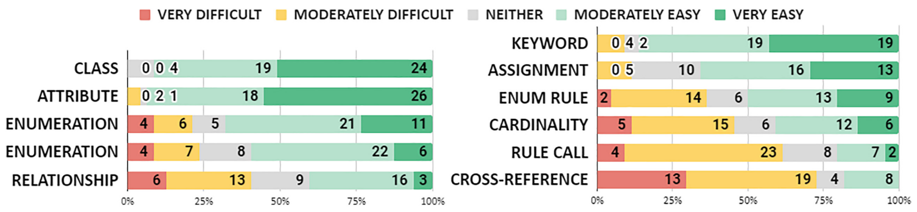


Fig. 5. The perceived difficulty of the metamodel primitives (left) and the grammar primitives (right) for the 2022 edition

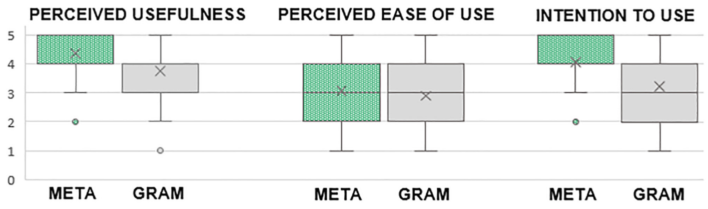


Fig. 6. The results of the Method Evaluation Model questionnaire (2022 edition)

that the participants found evolving the artefact useful and easy to use, and that they intend to use the artefact in the future. As for usefulness, the metamodel is perceived as more useful than the grammar, although they are generally both perceived as useful. Interestingly, the perceived ease of use for the grammar and metamodel is similar. Almost half of the participants expressed that they do not intend to use Xtext grammars in the future.

5 Discussion, Validity and Conclusions

Discussion. When reflecting on the evolution process, the participants expressed the need for longer training sessions with more guidance and literature they could refer back to. They expressed that they found using and evolving the metamodel easier due to their prior training in method engineering. Based on the numerous suggestions by students we conclude that training is a crucial part of DSML evolution. We improved the training in 2022, hoping that it would improve the results. Generally, the results are better in the second edition of the study. In the first year, the training did not include the openESEA DSML, instead, the participants were trained on DSMLs in different domains. The participants were only confronted with the openESEA DSML during the empirical study. In the second year, the training included an explanation of each of the openESEA metamodel and grammar primitives. Additionally, in the second year, the students were provided with self-paced learning material on Xtext.

We use completeness to refer to task completeness of the evolution activities; that is, the extent to which the team evolving the DSML successfully made all the changes that were required to implement the requirements in each DSML artefact, as is typical in usability studies [36]. Therefore, completeness should not be understood in the same way as it is defined by SEQUAL [21]. Task completeness aggregates both dimensions of SEQUAL's semantic quality: completeness and validity. Moreover, we solely focus on DSML evolution based on functional requirements. To limit the scope we have purposely left out quality requirements.

Validity. Regarding construct validity, we use the MEM questionnaire to analyse the participants' perceptions, since the MEM has been validated thoroughly [27]. We analyse incompletions in DSML evolution by calculating completeness of the DSML evolution activities. Variables, such as efficiency and consistency were left for future studies. Concerning external validity, we expect that our results can be generalised outside the setting of our study since we present our results in the context of commonly used UML and Xtext primitives. However, we acknowledge that there are other factors that influence the DSML evolution process, such as the usability of the DSML [1], or the participants' knowledge of the ESEA domain. Although we have opted for an approach that seems intuitive to us (i.e. first evolving the metamodel, then involving the grammar), and we tried to ensure human readability in the grammar, we cannot assess the usability of our DSML. Thus, we cannot identify the effect of the usability of our DSML on the incompletions witnessed in our study. Given that the study only assesses task completeness, there might be a threat to content validity. Nonetheless, we deem our results valuable in the evolution of domain-specific modelling languages, to prevent complications during MDE projects. To mitigate another content validity threat, we built upon an existing reengineering framework, and used the conceptual model evolution traces defined in [35]; this is reflected in 3, where we show model element additions, updates and deletions with green, yellow and red, respectively.

Next Steps. Regarding the evolution of the DSML, our next steps involve extending the language to support more aspects of ethical, social and environmental accounting. We would, for instance, like to include classes and primitives that allow for auditing and visualising accounts. With regards to our research on DSML evolution, the next steps could entail measuring additional variables such as the efficiency of DSML evolution and studying which cognitive aspects are related to the various artefacts.

Conclusion. In summary, we presented an empirical study that analyses which incompletions are prevalent in DSML evolution activities. Our results reveal that the completeness of the metamodel evolution activity is about 1.25 times higher than it is for the grammar. The metamodelling primitives that are more likely to cause problems are relationships and enumerations. With respect to the Xtext grammars, most incompletions are localised in rule calls, cross references and cardinalities. We also found that the metamodel is perceived as an important artefact to understanding and evolving the DSML. Additional training with emphasis on the problematic elements may result in fewer incompletions in DSML evolutions.

References

1. Albuquerque, D., Cafeo, B., Garcia, A., Barbosa, S., Abrahão, S., Ribeiro, A.: Quantifying usability of domain-specific languages: an empirical study on software maintenance. JSS **101**, 245–259 (2015)
2. Aschauer, T., Dauenhauer, G., Pree, W.: A modeling language's evolution driven by tight interaction between academia and industry. In: ICSE, IEEE (2010)
3. Behrens, H., et al.: Xtext user guide. Eclipse Foundation (2010)
4. Cazzola, W., Poletti, D.: DSL evolution through composition. In: RAM-SE (2010)
5. Ceh, I., Crepinšek, M., Kosar, T., Mernik, M.: Ontology driven development of domain-specific languages. Comput. Sci. Inf. Syst. **8**(2), 317–342 (2011)
6. Cook, S., Jones, G., Kent, S., Wills, A.C.: Domain-Specific Development with Visual Studio dsl tools. Pearson, London (2007)
7. De Geest, G., Vermolen, S., Van Deursen, A., Visser, E.: Generating version convertors for domain-specific languages. In: WCRE, pp. 197–201. IEEE (2008)
8. España, S., Bik, N., Overbeek, S.: Model-driven engineering support for social and environmental accounting. In: RCIS, pp. 1–12. IEEE (2019)
9. España, S., Ramautar, V., Overbeek, S., Derikx, T.: Model-driven production of data-centric infographics: an application to the impact measurement domain. In: Guizzardi, R., Ralyté, J., Franch, X. (eds.) Research Challenges in Information Science. RCIS 2022. Lecture Notes in Business Information Processing, vol. 446, pp 477–494. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05760-1_28
10. Falazi, G., Breitenbücher, U., Daniel, F., Lamparelli, A., Leymann, F., Yussupov, V.: Smart contract invocation protocol (SCIP): a protocol for the uniform integration of heterogeneous blockchain smart contracts. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 134–149. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_9
11. Frank, U.: Domain-specific modeling languages: requirements analysis and design guidelines. In: Reinhartz-Berger, I., Sturm, A., Clark, T., Cohen, S., Bettin, J. (eds.) Domain Engineering, pp. 133–157. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-36654-3_6
12. Gray, R., Adams, C.A., Owen, D.: Accountability, social responsibility and sustainability. Pearson, London (2014)
13. Gronback, R.C.: Eclipse Modeling Project: a Domain-Specific Language (DSL) Toolkit. Pearson Education, London (2009)
14. Hailpern, B., Tarr, P.: Model-driven development: the good, the bad, and the ugly. IBM Syst. J. **45**(3), 451–461 (2006)
15. Herrmannsdoerfer, M., Ratiu, D., Wachsmuth, G.: Language evolution in practice: the history of GMF. In: van den Brand, M., Gašević, D., Gray, J. (eds.) SLE 2009. LNCS, vol. 5969, pp. 3–22. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12107-4_3
16. Izquierdo, J.L.C., Cabot, J.: Community-driven language development. In: MISE, pp. 29–35. IEEE (2012)
17. Jazayeri, B., Schwichtenberg, S., Küster, J., Zimmermann, O., Engels, G.: Modeling and analyzing architectural diversity of open platforms. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 36–53. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_3
18. Juergens, E., Pizka, M.: The language evolver lever-tool demonstration-. Electron. Notes Theor. Comput. Sci. **164**(2), 55–60 (2006)

19. Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: a systematic mapping study. IST **71**, 77–91 (2016)
20. Kosar, T., et al.: Comparing general-purpose and domain-specific languages: an empirical study. Comput. Sci. Inf. Syst. **7**(2), 247–264 (2010)
21. Krogstie, J.: Model-based Development and Evolution of Information Systems: A Quality Approach. Springer, New York (2012). https://doi.org/10.1007/978-1-4471-2936-3
22. Kulkarni, V., Reddy, S., Clark, T.: Advanced Digital Architectures for Model-Driven Adaptive Enterprises. IGI Global, Hershey (2020)
23. Lientz, B.P., Swanson, E.B.: Software Maintenance Management. AW, Boston (1980)
24. Mayer, P., Schroeder, A.: Towards automated cross-language refactorings between Java and DSLs used by Java frameworks. In: WRT (2013)
25. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. CSUR **37**(4), 316–344 (2005)
26. Meyers, B., Vangheluwe, H.: A framework for evolution of modelling languages. Sci. Comput. Program. **76**(12), 1223–1246 (2011)
27. Moody, D.L.: The method evaluation model: a theoretical model for validating information systems design methods (2003)
28. Nikolov, N., Rossini, A., Kritikos, K.: Integration of DSLs and migration of models: a case study in the cloud computing domain. Procedia CS **68**, 53–66 (2015)
29. OMG: Unified Modeling Language, Version 2.5.1 (2017)
30. Pastor, Ó., España, S.: Full model-driven practice: from requirements to code generation. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 701–702. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31095-9_48
31. Phillips, M., Lu, J.: A quick look at NVivo. J. Electron. Resour, Librariansh (2018)
32. Ramautar, V., España, S.: Evolution of the openESEA DSL. Technical Report (2022). https://doi.org/10.17632/2xjbs6x6bp.1
33. Ramautar, V., España, S.: Managing the complexity in ethical, social and environmental accounting: engineering and evaluating a modelling language. In: ManComp (2022)
34. Rieger, C., Westerkamp, M., Kuchen, H.: Challenges and opportunities of modularizing textual domain-specific languages. In: MODELSWARD, pp. 387–395 (2018)
35. Ruiz, M., España, S., Pastor, Ó., Gonz, A., et al.: Supporting organisational evolution by means of model-driven reengineering frameworks. In: IEEE 7th International Conference on Research Challenges in Information Science (RCIS), pp. 1–10. IEEE (2013)
36. Seffah, A., Kececi, N., Donyaee, M.: Quim: a framework for quantifying usability metrics in software quality models. In: APSEC, IEEE (2001)
37. Sendall, S., Kozaczynski, W.: Model transformation: the heart and soul of model-driven software development. IEEE Softw. **20**(5), 42–45 (2003)
38. Strembeck, M., Zdun, U.: An approach for the systematic development of domain-specific languages. Softw. Pract. Experience **39**(15), 1253–1292 (2009)
39. Thanhofer-Pilisch, J., Lang, A., Vierhauser, M., Rabiser, R.: A systematic mapping study on DSL evolution. In: Euromicro, pp. 149–156. IEEE (2017)
40. Van Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal question metric (GQM) approach. Encyclopedia of software engineering (2002)
41. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, New York (2012). https://doi.org/10.1007/978-3-642-29044-2