Towards an Agile and Ontology-Aided Modeling Environment for DSML Adaptation

Emanuele Laurenzi[1,2,3,*], Knut Hinkelmann[1,3], Stefano Izzo[1], Ulrich Reimer[2], and Alta van der Merwe[3]

[1] FHNW University of Applied Sciences and Arts Northwestern Switzerland, Riggenbachstrasse 16, 4600 Olten, Switzerland
{emanuele.laurenzi,knut.hinkelmann}@fhnw.ch, stefano.izzo.fhnw@gmail.com
[2] IPM-FHSG, Institute of Information and Process Management, University of Applied Sciences St. Gallen, Rosenbergstrasse, 59, 9001 St. Gallen, Switzerland
ulrich.reimer@fhsg.ch
[3] Department of Informatics, University of Pretoria, Lynnwood Rd, Pretoria 0083, South Africa
alta.vdm@up.ac.za

Abstract. The advent of digitalization exposes enterprises to an ongo-ing transformation with the challenge to quickly capture relevant aspects of changes. This brings the demand to create or adapt domain-specific modeling languages (DSMLs) efficiently and in a timely manner, which, on the contrary, is a complex and time-consuming engineering task. This is not just due to the required high expertise in both knowledge engineering and targeted domain. It is also due to the sequential approach that still characterizes the accommodation of new requirements in modeling language engineering. In this paper we present a DSML adaptation approach where agility is fostered by merging engineering phases in a single modeling environment. This is supported by ontology concepts, which are tightly coupled with DSML constructs. Hence, a modeling environment is being developed that enables a modeling language to be adapted on-the-fly. An initial set of operators is presented for the rapid and efficient adaptation of both syntax and semantics of modeling languages. The approach allows modeling languages to be quickly released for usage.

Keywords: Agile modeling environment
Domain-specific adaptation
Enterprise modeling language engineering
Ontology-aided modeling environment Domain-specific modeling language

1 Introduction

With the advent of digitalization, model-driven approaches are receiving more attention in Enterprise Modeling [1]. Enterprises are exposed to an ongoing

transformation with the challenge to quickly and efficiently capture relevant aspects of changes. Burlton et al. [2], in their Business Agility Manifesto argue that it is not sufficient to overcome this challenge with a faster software development - "once operational, such software is likely to prove difficult to continuously and rapidly change without unintended consequences". The current *Knowledge Age* has shifted the purpose of modeling from software development to the creation of knowledge bases through models. Hence, models are becoming more and more means of representing relevant knowledge about business models, business processes, organization structure or resources, which can be used for automation and operations. These models have the ultimate objective to support decision makers. For instance, Enterprise Architecture models support decision makers in business transformation [3]. Models are built using modeling languages, which in turns should enable accommodating evolving requirements, ideally in a way that can be easily understood by experts and stakeholders within a targeted domain. However, existing modeling languages might not be expressive and concise enough to address a specific application domain and therefore may need to be adapted, i.e. extend modeling constructs (i.e. concepts and/or relations), remove unnecessary ones, integrating constructs from different languages or assigning predefined value types as well as concrete values. In this context model-driven domain-specific modeling language adaptation is still a time consuming and complex engineering effort. Namely, it requires numerous iterative phases until a modeling language is rolled out. Some (or all) phases most of the times are still performed sequentially, recalling the rigid waterfall methodology of software engineering. In particular, a modeling language can be validated only after it is implemented and, in turn, the latter starts after the design phase has taken place. That means, some modeling requirements conceptualized in early phases may become outdated while validating the modeling language in the final phases. Although some agile modeling engineering approaches were introduced to allow intertwining phases (e.g. in [1]), a modeling requirement still needs to go through all the engineering phases sequentially in order to be solidly embedded in the modeling language. Additionally, the complexity of domain-specific adaptation goes at the expenses of the duration of the engineering phases. This complexity mainly resides on the lack of development support, i.e. scarce availability of guidelines and best practices [4] as well as the required domain expertise that the knowledge engineer (i.e. developer of the modeling language) should have but rarely has [5].

This engineering effort reclaims supportive modeling approaches that fosters agility along the engineering life-cycle. In contribution to that, this work proposes an agile and ontology-aided modeling environment for domain-specific modeling language adaptation, which enables knowledge engineers accommodating new requirements in a timely and efficient manner. The rest of the paper is structured as follows: Sect. 2 presents the theoretical background supporting the derivation of requirements for an agile modeling environment allowing for domain-specific modeling adaptation. Next, Sect. 3 presents the main idea of the modeling environment and motivates the adoption of ontologies as means to

support agility. This section also address the previously derived requirements, and the first set of operators for the domain-specific adaptation is introduced. Finally, Sect. 4 presents the validation of our approach (1) by developing a modeling environment that meets the requirements discussed in the previous section and (2) by implementing the introduced operators on our modeling environment, which are in turn validated in a research project use case.

2 Background

Domain-Specific Adaptation: With the need to address particular application domains, existing modeling languages might not be expressive and concise enough to be adopted. This can lead to the need of adapting existing modeling languages through a domain-specific conceptualization [6,7]. In result, a domain-specific modeling language (DSML) [8,9] is created. Conversely to a general-purpose modeling language (GPML), concepts of a DSML are tailored to a specific domain and are represented by graphical notations familiar to the user of the models. Hence, complexity shifts from the model (i.e. level one) to the meta-model (i.e. level two) to ease the design and understanding of models by domain experts or modelers. Additionally, DSMLs foster the creation of uniform models, and thus support producing quality models. Developing a DSML can be done from scratch or through domain-specific adaptation of existing modeling languages. The latter approach provides the benefit of considering established experience and lessons learned from existing modeling languages or notations. This in turn fosters the reusability (total or partial) of the modeling constructs within the modeling community or across projects. The domain-specific adaptation includes the possibility of (a) introducing new modeling constructs, (b) removing, modifying, replacing existing ones, or borrowing constructs from other application domains and provided with a new semantics. (c) integrating modeling constructs that belong to different modeling languages. In [1], these actions are defined as extensibility, adaptability and integrity, respectively.

Benefits of a domain-specific adaptation come, however, at the expenses of a higher engineering effort for the knowledge engineer, who has to embed domain aspects in the modeling language. Namely, she/he is demanded to understand both (1) the semantics of the modeling language(s) to adapt and (2) the domain knowledge that needs to be covered by the adapted language. This requires a significant experience in modeling and numerous meetings with domain experts [10] to gain domain knowledge. A further complexity lies in (3) expressing the modeling language on right level of abstraction, which leads to complex trade-off decisions between productivity and re-usability of modeling elements [5].

Providing support to the knowledge engineer to perform a domain-specific modeling language adaptation is the first requirement of our modeling environment (*Requirement(1)*).

Meta-Modeling as Means for Domain-Specific Adaptation: In model-driven approaches, the way DSMLs are defined is often rooted to the meta-modeling hierarchy [11]. Meta-modeling is a model-driven technique adopted to

ease the development of a modeling language and, thus applicable for DSMLs too. It is a common practice to specify a modeling language in Level two (see Fig. 1). For example, standard modeling languages like ArchiMate, BPMN, CMMN, DMN are typically modeled as UML class diagrams in Level two. Thus, Level two (L2) contains the meta-model, which defines the modeling language to create models in Level one (L1) (see the two levels in Fig. 1).

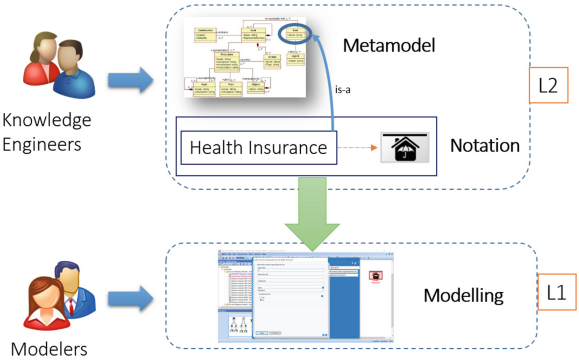

Fig. 1. Meta-model in level two (L2) and Model in level one (L1) [12]

A modeling language comprises *abstract syntax*, *notation* and *semantics* [13]. Abstract syntax corresponds to the class hierarchy of modeling constructs, which consist of modeling elements and properties (i.e. relations and attributes). Modeling constructs are typically expressed through notations (e.g. graphical or textual), also known as concrete syntax, which should be cognitively adequate to ensure user's understanding of models [12]. The semantics reflects the meaning of the syntactic elements of a modeling language. It can be divided into *structural* and *behavioral* semantics. While structural semantics reverts to the meaning of the class-instance representation, the behavioral one relates to model execution. In this paper we use the term semantics to refer to the structural semantics only. This can be expressed formally (i.e. through mathematics or ontologies) or informally (i.e. through natural language). The abstract syntax of a language is commonly mapped to domain semantics concepts. Constraints (or rules) over the modeling constructs are needed, for example to specify cardinality restrictions or to express that two classes are disjoint. The modeling environment has to embrace a model-driven approach to perform domain-specific modeling language adaptation (*Requirement(2)*).

Agility on the DSML Engineering Process: Developing a modeling language is an engineering task (see [14]), so it is the domain-specific modeling language adaptation. In modeling method engineering (which includes modeling languages) the OMiLab Lifecycle [15] defines the cycle of five phases: create, design, formalize, develop and deploy/validate (see Fig. 2). In general terms,

the engineering process that embraces the meta-modeling technique follows the iterative phases (a) conceptualizing the meta-model, (b) implementing it in a meta-modeling tool (e.g. ADOxx[1] or MetaEdit+[2]), and finally (c) validating the modeling language. The latter generates feedback, and determine language amendments. Obviously, the modeling language can be evaluated only after its meta-model is implemented. Hence, abstract syntax, constraints and graphical notations should be conceptualized, implemented and then used for evaluation purposes. This engineering process can be characterized by a sequential design approach, which resembles the waterfall-style approach of software development, where previous phases have to be accomplished before moving forward. To avoid this, the AMME framework [1] was introduced as an agile management approach that follows agile principles to engineer modeling methods. The OMiLab Lifecycle instantiated AMME by introducing feedback channels along the five phases (see black arrows in Fig. 2) with the objective to support the engineering process during the propagation and evolution of modeling requirements [16]. In result, various modeling languages were created following this agile modeling method engineering, e.g. [7,17].

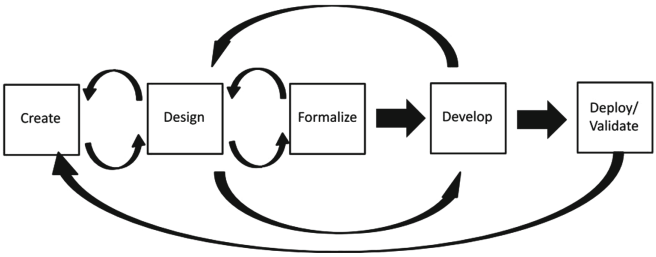

Fig. 2. The OMiLab lifecycle for modeling method engineering [16]

Although this approach allows intertwining most of the phases (e.g. create-design, design-formalize and design-develop) from the point of time a new requirement is accommodated until it is validated it has to go through all or most of the phases in a subsequent manner. For instance, if a new requirement arises while validating the modeling language, it needs to be captured and represented in the form of modeling requirement in the creation phase. Next, the latter is designed such that it fits other modeling constructs. At some point it might be formalized, then it is embedded in the prototype during the development phase and finally the modeling language is ready to be validated again. This sequential approach become problematic with the long duration of each engineering phase as the longer it takes the higher the risk to have outdated requirements. This risk can be avoided by eliminating as much as possible sequential phases in the case when new requirements arise. Hence, the modeling environment should foster

[1] https://www.adoxx.org/live/home.
[2] http://www.metacase.com/products.html.

agility by enabling the knowledge engineer to accommodate new requirements by avoiding as much as possible sequential phases (*Requirement(3)*).

Accommodation of Evolving Requirements: A DSML is subject to evolve over time. This is due to new modeling requirements or result from a better understanding of the domain [18]. Therefore, the knowledge engineer is continuously demanded to adapt the meta-model. Moreover, amendments of the meta-model are continuously required due to pitfalls related to inappropriate constraints, abstraction issues, or ambiguity of modeling elements. Also, decisions on whether to promote productivity rather than reusability of the modeling language [19] are subject to continuous changes. The more specific the concepts are (i.e. higher specificity level of the modeling language and thus higher productivity), the lower the possibility to reuse the modeling language across domains or even in different areas, processes or projects of the same domain. Within this context, it becomes even more relevant to support the knowledge engineer with a modeling environment that quickly allows accommodating modeling requirements, ideally on-the-fly. Modeling tools like Visual Paradigm[3] address this challenge by implementing the UML mechanisms *stereotype* and *tagged values*. Hence, the user is enabled on-the-fly to customize modeling constructs. However, adapting a modeling language may lead to a change in the semantics. For instance, specializing an existing modeling construct requires a new interpretation of the new inserted modeling construct. The same applies when editing an existing modeling construct or adding a new one. The new interpretation has to be specified as the semantics of the modeling construct to avoid ambiguous interpretation and non-sense constructs. It is also important that the adaptation of the modeling language does not lead to side effects. For example, adding or removing modeling constructs should not lead to unwanted results or consistency issues. The modeling environment should foster both agility and model quality by enabling the knowledge engineer to accommodate new requirements on-the-fly and efficiently (*Requirement(4)*).

3 The Agile and Ontology-Aided Modeling Environment

Merging Engineering Phases in a Single Integrated Modeling Environment: Different engineering phases can address different expertise. For instance, while conceptualization and implementation are tasks of knowledge engineers, validation often requires the involvement of modelers. This often leads to the adoption of different tools: one for the development of a DSML (occurring in Level 2) and one for using and validating the language and thus creating models (occurring in Level 1). The adoption of two separate tools reflects per se an implementation of the sequential engineering process introduced in Sect. 2. In order to foster agility by avoiding sequential phases (see *Requirement(3)*), we propose a single environment which integrates Level one with Level two. This implies that

[3] https://www.visual-paradigm.com/.

the different engineering phases are being merged in the same environment and performed in parallel. The integrated environment places knowledge engineers and modelers in the center of our approach (Fig. 3) by creating conditions for the two roles to provide feedback to each other in a timely manner. *Requirement(1)* and *Requirement(2)* are full-filled as domain-specific modeling language adaptation is allowed within model-driven context. Section 4 shows how the domain-specific adaptation takes place in the modeling environment. Performing the engineering phases in parallel means that while the knowledge engineer adapts modeling languages, the implementation and validation can occur at the same time. For this, a formalization of the knowledge that results from all changes that occur in the modeling language is required. Hence, the formalization has the purpose to automate the execution of engineering phases in parallel. For instance, assuming a new modeling element is inserted (see "Acute Hospital" concept in Fig. 3), its formalization and validation should occur automatically. According to Bork and Fill in [4] a formal specification is necessary to provide unambiguous understanding of models and to foster the interoperability between different computer systems. The same can be applied to modeling languages with the advantage that a formal specification of a modeling language can be automatically propagated on models. A well-known approach to formally define semantics of modeling languages and models is by means of the semantic lifting [20]. Modeling elements and their instances are associated with ontology concepts, which are represented in logic-based languages [17,21]. The problem of this approach, however, is to ensure consistency between the modeling language and related ontology concepts. If a change occurs in the language or any of the models, the ontology should be adapted accordingly. To avoid this problem the modeling environment is fully ontology-based. Namely, modeling constructs are formally defined through ontology concepts and tightly coupled with the respective graphical notations, which avoids the consistency problems caused by the semantic lifting. Knowledge engineers and modelers can rely on customizable graphical notations for increasing clarity of models. Having a modeling language that is ontology-based allows to build ontology-based models. Hence, both modeling languages and models can then be used for reasoning services (like in [22]). This fulfills *Requirement(4)* as new requirements can be accommodated on-the-fly and efficiently.

This approach builds on the semantic meta-model idea introduced in [12]. We take a step forward by distinguishing between the *Domain Ontology* and the *Language Ontology*.

Domain Ontology: The Domain Ontology refers to what Atkinson [23] calls the Ontological meta-modeling View. The Domain Ontology contains classes, properties and instances that describe a domain of discourse. For example, in the health domain there are concepts like patient, disease, physician or hospital, which are structured in a class hierarchy (see right-hand side of Fig. 3). The domain ontology can be contain standards to represent domain knowledge like the International Classification of Functioning, Disability and Health (ICF)

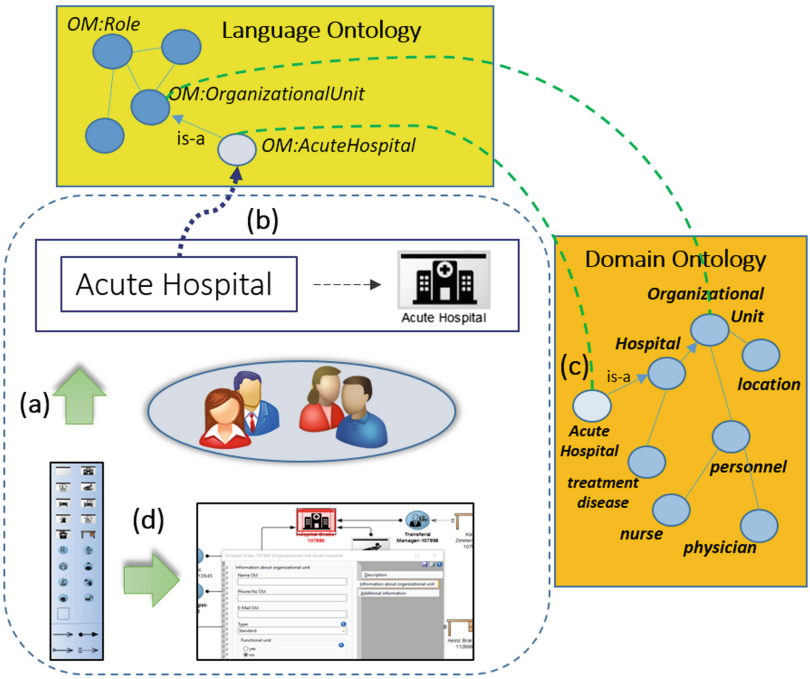

Fig. 3. Conceptualization of the Agile and Ontology-aided Modeling Environment

ontology[4]. Concepts in the Domain Ontology represent the semantics of language constructs, which intends to be independent from a particular modeling language. For instance, the semantics of a modeling construct like "Gateway" from BPMN can be the same as the "Sentry" from CMMN, i.e. both can express the same condition. For this, concepts of the Domain Ontology are mapped to modeling constructs that reside in the Language Ontology.

Language Ontology: The Language Ontology refers to what Atkinson [23] calls the Linguistic meta-modeling View. The Language Ontology contains classes, properties and instances that describe the syntax elements of a modeling language, i.e. modeling elements and modeling relations with respective taxonomy, object properties can occur between the modeling elements and modeling relations (e.g. for the specification of source and target from a modeling relation to a modeling element). Each modeling element and relation is linked to a graphical notation through a data type property (i.e. attribute). Instances can represent types of modeling constructs, e.g. see task types like user task, service task etc. in [24]. The Language Ontology can contain one or more modeling languages, which are separate from each other or integrated [25]. The Language Ontology supports the adaptation of the modeling language (see vertical arrow of Fig. 3). In the modeling environment, new modeling elements can be added for

[4] https://bioportal.bioontology.org/ontologies/ICF.

existing domain concepts. As an example, we assume that the palette contains already the modeling elements of the Organizational Model. From the palette (see step (a) in Fig. 3), the knowledge engineer can specialize the concept organizational unit to an acute hospital concept (see step (b) in Fig. 3). If this concept is not yet defined in the Domain Ontology, it is added there as a specialization of the concept hospital (see step (c) in Fig. 3).

Operators for Adapting a Modeling Language: The adaptation of a modeling language can be done by applying a set of operators on the Language Ontology. In order to determine an initial set of operators, we build on lessons learned from the work of recent research projects, Patient-Radar [6,26] and CloudSocket[5] [27,28]. Thus, the following set of operators were derived to be performed on the Language Ontology, which implies a theoretical foundation based on ontology formalism. **Operator 1**. Create sub-class: It is applied on modeling elements and modeling relations to create new modeling elements and new modeling relations. This operator is also applied to integrate modeling elements (classes) from different modeling languages. **Operator 2**. Delete sub-class: It is applied on modeling elements and modeling relations to remove unneeded modeling elements and modeling relations from the modeling language. **Operator 3**. Create relation: It connects modeling elements and modeling relations to the related Domain Ontology concept. **Operator 4**. Update relation: It is applied on modeling relations as it allows updating existing connections between modeling elements/relations and the related Domain Ontology concepts. **Operator 5**. Delete relation: It allows deleting existing connections between modeling elements/relations and the related Domain Ontology concepts. **Operator 6**. Create attribute: It allows adding new attributes to modeling elements and modeling relations. **Operator 7**. Update attribute: It allows updating existing attributes. **Operator 8**. Delete attribute: It allows deleting existing attributes. **Operator 9**. Assign attribute type: It allows assigning value types String, Integer, Boolean to attributes of modeling elements. **Operator 10**. Update attribute types: It allows updating types that are assigned to attributes of modeling elements.

4 Validation

The validation took place (a) by developing the modeling environment such that the four requirements introduced in Sect. 2 were met as discussed in Sect. 3 and (b) by implementing the derived list of operators on the modeling environment. The operators were also validated by applying them on a use case of the Patient-Radar research project. The BPMN modeling language was adapted on-the-fly with modeling elements from the patient transferal domain [6]. Figure 4 shows the screen-shot of **Operators 1 and 2** as illustration. Namely, a subclass of the User Task is created by right-clicking on the User Tasks element in the palette.

[5] https://site.cloudsocket.eu/.

The pop-up window in screen-shot number 2 allows the user to (a) assign a name to the new element, (b) specify whether the graphical notation element must be shown in the palette, (c) assign the image for the graphical appearance and (d) map the element to its semantics in the Domain Ontology. The new element is then stored in the Language Ontology as a new class. The screen-shot number 3 of Fig. 4 shows the hierarchy of the modeling elements. The new added element "Prepare KoGu" is shown in the last tier as a sub-concept of "User Task". **Operators from 3 to 8:** By right-clicking on an element in the palette, a context menu is shown, which enables the user to edit the element. A pop-up window provides the possibility to add, modify or delete a relation or an attribute. **Operators 9 and 10:** By right-clicking on an element in the palette, the user can select "create new property". For a new data type property the user is able to assign attribute types as well as assigning predefined values.

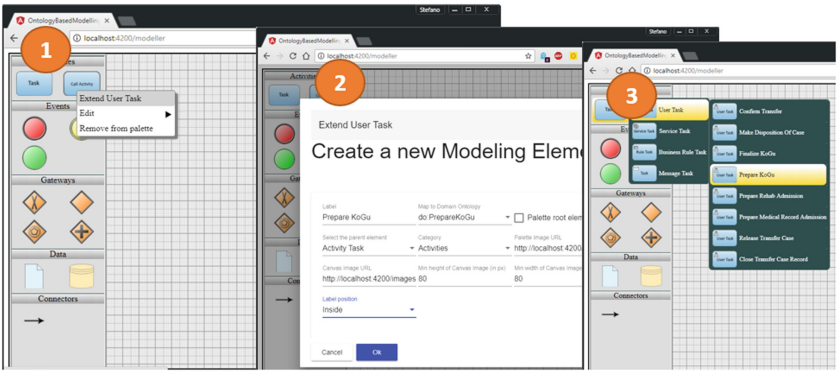

Fig. 4. Operators 1 and 2

5 Conclusion

In this paper the first set of requirements for an agile modeling environment was derived. This emerged from the need to rapidly and efficiently create and adapt domain-specific modeling languages in the context of Enterprise Modeling. Hence, we addressed the main hinder: the subsequent modeling language engineering phases, which prevents a domain-specific modeling language to be quickly rolled out. The new approach allows to perform engineering phases such as conceptualization, implementation and validation all at once in a single modeling environment. The latter integrates Level two and Level one, where the meta-model and model are created, respectively. Thus, the knowledge engineer is enabled to adapt the meta-model and create models from the same modeling environment. The grounding of modeling contructs with ontology concepts supports the agile approach (i.e. on-the-fly accommodation of new requirements) by ensuring that operations on the modeling language are reflected in the Language Ontology. A first set of operators to apply on the Language Ontology

was derived. This allowed the quick propagation of a modeling requirements: from its implementation, to its formal representation, which makes it ready to be validated by applying reasoning services on the Language Ontology. The formalism of the ontology also removes ambiguity on the meaning of language constructs. These benefits are propagated to the models which are built with the formally grounded modeling language. A distinction between Language Ontology and Domain Ontology was also introduced. While the Language Ontology is strictly related to the structure of modeling languages, the Domain Ontology contains the (language-independent) semantics of language constructs. Hence, the Domain Ontology is highly portable as it can be mapped to various modeling languages. The approach was validated by developing the modeling environment with respect to the discussed requirements. Additionally, the set of operators was implemented on the modeling environment and validated in a project's use case. Future work goes towards the improvement of our prototype, which comprises an extension of set of operators as well as applying reasoning services such as consistency checking on both Language and Domain Ontology.

References

1. Karagiannis, D.: Agile modeling method engineering. In: Proceedings of the 19th Panhellenic Conference on Informatics - PCI 2015, pp. 5–10. ACM Press, New York (2015)
2. Burlton, R.T., Ross, R.G., Zachman, J.A.: The Business Agility Manifesto. https://busagilitymanifesto.org
3. Zachman, J.A.: The Concise Definition of The Zachman Framework by: John A. Zachman (2008). https://www.zachman.com/about-the-zachman-framework
4. Bork, D., Fill, H.G.: Formal aspects of enterprise modeling methods: a comparison framework. In: 2014 47th Hawaii International Conference on System Sciences, pp. 3400–3409. IEEE (2014)
5. 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, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36654-3_6
6. Laurenzi, E., Hinkelmann, K., Reimer, U., van der Merwe, A., Sibold, P., Endl, R.: DSML4PTM - a domain-specific modelling language for patient transferal management. In: ICEIS 2017 - Proceedings of the 19th International Conference on Enterprise Information Systems, Porto, Portugal, pp. 520–531 (2017)
7. Karagiannis, D., Buchmann, R.A., Burzynski, P., Reimer, U., Walch, M.: Fundamental conceptual modeling languages in OMiLAB. Domain-Specific Conceptual Modeling, pp. 3–30. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_1
8. Clark, T., van den Brand, M., Combemale, B., Rumpe, B.: Conceptual model of the globalization for domain-specific languages. In: Combemale, B., Cheng, B., France, R., Jézéquel, J.M., Rumpe, B. (eds.) Globalizing Domain-Specific Languages. LNCS, vol. 9400, pp. 7–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26172-0_2
9. van Deursen, A., Klint, P., Visser, J.: Domain-specific languages: an annotated bibliography. SIGPLAN Not. **35**(6), 26–36 (2000)

10. Nikles, S., Brander, S.: Separating Conceptual and Visual Aspects in Meta-Modelling, pp. 90–94 (2009)
11. Strahringer, S.: Metamodellierung als Instrument des Methodenvergleichs: Eine Evaluierung am Beispiel objektorientierter Analysenmethoden. Publications of Darmstadt Technical University, Institute for Business Studies (BWL) (1996)
12. Hinkelmann, K., Laurenzi, E., Martin, A., Thönssen, B.: Ontology-based metamodeling. In: Dornberger, R. (ed.) Business Information Systems and Technology 4.0. SSDC, vol. 141, pp. 177–194. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74322-6_12
13. Karagiannis, D., Kühn, H.: Metamodelling platforms. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) EC-Web 2002. LNCS, vol. 2455, pp. 182–182. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45705-4_19
14. Hölldobler, K., Roth, A., Rumpe, B., Wortmann, A.: Advances in modeling language engineering. In: Ouhammou, Y., Ivanovic, M., Abelló, A., Bellatreche, L. (eds.) MEDI 2017. LNCS, vol. 10563, pp. 3–17. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66854-3_1
15. Laboratory Open Models - OMILab: Idea and Objectives (2015)
16. Efendioglu, N., Woitsch, R., Karagiannis, D.: Modelling method design. In: Proceedings of the 17th International Conference on Information Integration and Web-based Applications and Services - iiWAS 2015, pp. 1–10. ACM Press, New York (2015)
17. Buchmann, R.A.: Modeling product-service systems for the internet of things: the ComVantage method. Domain-Specific Conceptual Modeling, pp. 417–437. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_19
18. Götzinger, D., Miron, E.-T., Staffel, F.: OMiLAB: an open collaborative environment for modeling method engineering. Domain-Specific Conceptual Modeling, pp. 55–76. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39417-6_3
19. Frank, U.: Multilevel modeling toward a new paradigm of conceptual modeling and information systems design (2013)
20. Azzini, A., Braghin, C., Damiani, E., Zavatarelli, F.: Using semantic lifting for improving process mining: a data loss prevention system case study (2013)
21. Emmenegger, S., Hinkelmann, K., Laurenzi, E., Martin, A., Thönssen, B., Witschel, H.F., Zhang, C.: An ontology-based and case-based reasoning supported workplace learning approach. In: Hammoudi, S., Pires, L.F., Selic, B., Desfray, P. (eds.) MODELSWARD 2016. CCIS, vol. 692, pp. 333–354. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66302-9_17
22. Walter, T., Parreiras, F.S., Staab, S.: An ontology-based framework for domain-specific modeling. Softw. Syst. Model. **13**(1), 83–108 (2014)
23. Atkinson, C., Kuhne, T.: Model-driven development: a metamodeling foundation. IEEE Softw. **20**(5), 36–41 (2003)
24. Rospocher, M., Ghidini, C., Serafini, L.: An ontology for the business process modelling notation. In: 8th International Conference on Formal Ontology in Information Systems (2014)
25. Hinkelmann, K., Pierfranceschi, A., Laurenzi, E.: The knowledge work designer-modelling process logic and business logic. In: Proceedings - Series of the Gesellschaft fur Informatik (GI). Lecture Notes in Informatics (LNI), vol. P255 (2016)
26. Reimer, U., Laurenzi, E.: Creating and maintaining a collaboration platform via domain-specific reference modelling. In: EChallenges e-2014 Conference: 29–30 October 2014, Belfast, Ireland, pp. 1–9. IEEE (2014)

27. Hinkelmann, K., Laurenzi, E., Lammel, B., Kurjakovic, S., Woitsch, R.: A semantically-enhanced modelling environment for business process as a service. In: 2016 4th International Conference on Enterprise Systems (ES), pp. 143–152. IEEE (2016)
28. Kritikos, K., Laurenzi, E., Hinkelmann, K.: Towards business-to-IT alignment in the cloud. In: Mann, Z.Á., Stolz, V. (eds.) ESOCC 2017. CCIS, vol. 824, pp. 35–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79090-9_3