A Metamodeling Approach to Teaching Conceptual Modeling "at Large"

Ana-Maria Ghiran
Babeș-Bolyai University,
Business Informatics Research Center
Cluj-Napoca, Romania *anamaria.ghiran@econ.ubbcluj.ro*

Cristina-Claudia Osman
Babeș-Bolyai University,
Business Informatics Research Center
Cluj-Napoca, Romania *cristina.osman@econ.ubbcluj.ro*

Robert Andrei Buchmann
Babeș-Bolyai University,
Business Informatics Research Center
Cluj-Napoca, Romania *robert.buchmann@econ.ubbcluj.ro*

Abstract

In the authors' university there is a challenge, with respect to Conceptual Modeling topics, of bridging the gap between bachelor-level studies and research work. At bachelor-level, Conceptual Modeling is subordinated to Software Engineering topics consequently making extensive use of software design standards. However, at doctoral level or in project-based work, modeling methods must be scientifically framed within wider-scoped paradigms - Design Science, Enterprise Modeling etc. In order to bridge this gap, we developed a teaching artifact to present Conceptual Modeling as a standalone discipline that can produce its own artifacts, driven by requirements in a variety of domains. The teaching artifact is an "agile modeling method" that is iteratively implemented by students. The key takeaway revelation for students is that a modeling language is a knowledge schema that can be tailored and migrated for specific purposes just like a database schema, to accommodate an application domain and its modeling requirements.

Keywords: Agile Modeling Method Engineering, Metamodeling, Teaching Conceptual Modeling, Resource Description Framework

1. Introduction

In this paper's context, "Conceptual Modeling at large" refers to a wider, domain-agnostic scope for the discipline and practice of (diagrammatic) Conceptual Modeling, compared to the popular perception that modeling is a "means to an end" ancillary to Software Engineering activities - a viewpoint that has been crystallized by standards supporting model-driven software engineering. The longstanding conference series on Conceptual Modeling (ER), although often presenting software engineering applications, generally manifests a wide scope - from philosophical foundations [1] to application areas beyond software engineering – e.g., Enterprise Architecture Management [2].

In the authors' university the students graduating bachelor programs in Business Information Systems or Computer Science come in contact with conceptual modeling topics as chapters subordinated to one or another of several software engineering disciplines (e.g., ER diagrams for database design, UML for documenting requirements analysis or system design). At the same time, *a wider and deeper understanding is required in research work* (project-based industry collaborations, doctoral and postdoctoral studies,

some master dissertations). Especially in industry collaborations domain-specificity tends to be a key requirement (e.g., in smart process automation), whereas in master or doctoral studies it is not sufficient to wear the hat of a "user" (who takes a modeling language for granted), but it is often necessary to be capable of design research in context – i.e., to expand standards, to hybridize standards or to develop agile modeling tools for specific experiments, contexts and requirements.

This gap in understanding and abstraction is comparable to the one between "database users" (who operate with data records, while taking a system or database design for granted) and those able to migrate or deploy their own database for evolving needs. While for database courses this gap is easily bridged (even during the same semester), it is not the same for conceptual modeling topics which are dispersed as "aspects" of other disciplines (database design, object-oriented programming, requirements engineering).

To solve this problem, in our master programs we aim to stimulate lateral thinking by showing that a modeling language is a *knowledge schema* that can be tailored and migrated in the same sense as a database schema, to ensure the semantic space that is required for a particular application domain or information system (i.e., its purpose is not limited to graphical documentation). Software Engineering is thus repositioned as an application domain that benefits from standards reflecting best practices and consensus; at the same time, a more general notion of "model value" is introduced - one that transcends application domains and follows the learning design recommendations published in [3].

This is what we call "teaching Conceptual Modeling at large" – it prepares students for an extended understanding of model value and of modeling methods as engineered artifacts, while establishing pragmatic relations between modeling and Design Science, Knowledge Management & Representation and Method Engineering.

A minimalist modeling method, to be described in this paper, was designed as a teaching artifact. It showcases to students: (i) a modeling method's building blocks through minimal examples that can be prototyped quickly in a modeling tool; and (ii) the conceptualization and implementation process based on the Agile Modeling Method Engineering framework [4] to enable the agile migration of a modeling prototype assuming evolving requirements. Therefore, the research question for which this artifact was developed is *How can we teach Conceptual Modeling in a way that reveals the "knowledge schema" nature of a modeling language, regardless of application domain and amenable to agile evolution?*

The remainder of the paper is organized as follows: Section 2 outlines the requirements for the proposed teaching artifact and provides an overview on the proposed solution. Section 3 presents the teaching artifact and method. Section 4 discusses observed outcomes. Section 5 comments on related works. The paper ends with conclusions.

2. Problem Statement and Solution Summary

By analyzing past project challenges (of the authors' own experience or those documented in [5]), several meta-requirements have been distilled for the proposed teaching artifact. These are synthesized in Table 1 together with their motivation (*Rationale*), paralleled by suggestions on how they were addressed (*Solution Approach*).

Traditionally, there has been a significant gap between these requirements and the dominant perception of students on diagrammatic Conceptual Modeling, as acquired during bachelor studies. Most of our master students come from Business Information Systems or Computer Science bachelor programs, with a minority (<10%) from Business Administration programs. Their experience with modeling is dominated by UML and ER diagrams (or BPMN, for a minority of Business Administration students) – however even these are employed strictly as graphical documentation for bachelor theses (typically with drawing tools providing diagramming "templates").

The value of models *as purposeful knowledge representation* is thus lost or diluted in the common use case of system documentation. We aim to reinforce that value by repositioning a modeling language as a knowledge schema that supports easily demonstrable pragmatic goals – model queries, rule-based mechanisms or interoperability to enable model-driven engineering. The graphical representation thus becomes only a superficial layer for rich

knowledge structures. By raising the abstraction level, modeling goals are attached to paradigms such as Design Science or Knowledge Management, thus suggesting theoretical frames for students who want to further pursue research on these topics. "Model quality" also comes into discussion, since it is linked to "purpose" and modeling method "requirements" (which may also evolve).

Table 1. Requirements on the proposed teaching artifact and means of addressing them

Requirement	Solution Approach	Rationale
***A**. To position Conceptual Modeling as a Design Science approach*	The notion of "modeling method" [6] (including a modeling language) is introduced as an artifact subjected to its own engineering process driven by "modeling requirements". The engineering process produces specific deliverables guided by situational requirements and evaluation criteria (derived from generic criteria proposed in [7]).	Students should gain the ability to create and customize modeling methods that are purposeful and situational, and to productively prototype them in the form of modeling tools.
***B.** To position Conceptual Modeling within the Knowledge Management paradigm*	Considering the existing works on revisiting Nonaka's knowledge conversion cycle [8] through the lens of Conceptual Modeling (e.g., [9]), modeling is presented as a means of Knowledge Externalization for which software engineering is one of many possible application areas. The "knowledge representation" quality of models is stressed by showcasing the ability of applying semantic queries on models, employing the Resource Description Framework (RDF) [10] as a model storage format.	Students should gain the ability of tailoring a modeling method for Knowledge Externalization purposes, to satisfy knowledge retrieval requirements (semantic queries or reasoning). A modeling language must be understood as a knowledge schema that can be migrated just like a database schema (with models taking on the role of "records").
***C.** To emphasize domain-specificity as a common situational requirement*	Inspired by the existing tradition in domain-specific language development and domain engineering [5,11], the approach highlights means of assimilating domain-specificity in modeling languages, or to apply such specificity to all building blocks of a modeling method.	Students should gain the ability of extending standard modeling languages or to create new ones, for domain-specific purposes and having in mind the knowledge retrieval goals (model queries and possible interoperability with model-driven systems).
***D.** To reveal the agility potential of modeling methods.*	The Agile Modeling Method Engineering [4] methodology is employed to evolve a modeling method through two iterations driven by additive requirements, with the help of fast prototyping (metamodeling) platforms.	Students should gain the ability to evolve a modeling tool according to changing requirements.

In addition to the requirements summarized in Table 1, several pragmatic goals have been distilled from feedback on earlier attempts to design our teaching artifact [**Error! Reference source not found.**]:

- **Minimalism:** The development of the modeling method should be demonstrable in 2 meetings x 3 hours each, plus an additional meeting for discussion (to map the hands-on experience on theoretical background provided by lectures, also suggesting potential extensions for student homework). The modeling language should introduce no more than 3 concepts (and necessary relations), thus reducing the complexity to a "Hello world" kind of demonstration – however one that *minimally touches all building blocks and is still aligned with the meta-requirements* in Table 1;
- **Intuitive constructivism:** Hands-on experience of students should clash against their dominant preconceptions in order to generate transformations across the educational objectives specified by Bloom's framework [13] - Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation. Students with heterogeneous background should be able to follow and replicate the demonstration;
- **Domain-specificity (without domain expertise)** should manifest in various aspects of the modeling method, suggesting further means of expanding this specificity. However, specificity should be minimal to avoid prerequisite domain expertise and distractions pertaining to domain understanding;

- **Generalizability** (only loose coupling to software engineering): The proposed artifact should be detached from model-driven software engineering standards (UML, ER). At the same time, it should be re-attachable to software engineering purposes through means that illustrate the "models are knowledge" principle (i.e., model queries instead of the tight coupling of code generation);
- **Familiarity:** Existing modeling experience should be leveraged through analogies (to e.g., activity modeling), further suggesting how students could develop their own customization of existing standards.

The teaching artifact introduced to satisfy these requirements is a modeling method that is demonstrated and evolved in two iterations together with students. Following this, students also implement as exam projects their own modeling method/tool, for their choice of domain and requirements (possibly extending the provided demonstration).

Therefore, the proposed artifact can be considered a modeling method "embryo" – sufficiently rich to showcase the core principles of Agile Modeling Method Engineering and, at the same time, open-ended for further expansion or generalization. The building blocks of this artifact are shown in Figure 1, each mapped to their enabling technologies (free versions for educational purposes are available for all of these):

a. ADOxx [14], a metamodeling platform on which modeling methods may be implemented, including notation, syntactic rules, semantics or model-driven functionality;
b. GraphDB [15], an RDF graph database server with ontological capabilities (to store models as knowledge graphs);
c. ADOxx-to-RDF [16], a plug-in for converting diagrammatic models in machine-readable RDF graphs, regardless of the modeling language used to create them; the graphs are stored in GraphDB to expose model content to semantic queries (SPARQL [17]) from arbitrary clients;
d. ADOScript, the built-in scripting language of ADOxx for implementing model-based functionality.

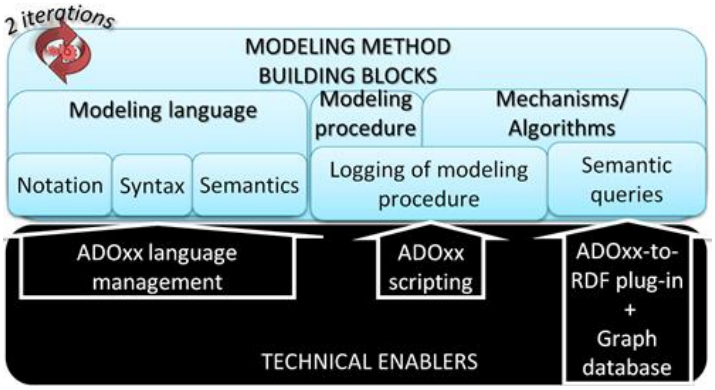

Fig. 1. Building blocks and enablers of the proposed teaching artifact

3. Methodology and Artifact

3.1. Application Domain and Method

The research method underlying this work is subordinated to the Design Science research paradigm [18] – i.e., we designed an artifact (a "modeling method") that is needed to improve a problem context - to enable master students to think not only as users of established modeling tools (taken for granted and bound to a modeling procedure), but also as creators guided by specific requirements in a narrow application domain.

Thus the artifact is iteratively built to defuse the fallacies and to satisfy the requirements formulated in Section 2, enabling new innovation competences in our Information Systems study programs, as well as an open-ended understanding of the benefits of Conceptual Modeling.

The application domain targeted by the teaching artifact is the Internet of Things, for which Conceptual Modeling can be used not only for traditional goals (e.g., system design), but also as a knowledge representation technique that is amenable for both analysis by humans and semantic processing by machines.

The proposed modeling method is introduced in relation to Knowledge Management requirements in a maintenance company. A knowledge base must accumulate diagnosing or repair procedures mapped on maintained devices and their diagnosing sensors. A modeling tool is required to build this knowledge base in diagrammatic form.

The teaching method is based on live demonstration of small implementation increments immediately followed by the same operation executed by students. The progress has a "gradual revealing" nature, with metamodeling theorization provided only at the end or in parallel lectures, to reflect back on experience and comparisons with known modeling tools or languages. Each modeling method building block is showcased by a minimal example enriched across two iterations in the two meetings. Exercising material is provided between the meetings (so that the second one is more productive).

The tool development method employed for hands-on exercising is a simplification of the Agile Modeling Method Engineering methodology. This is an iterative metamodeling approach where each iteration starts with the definition of domain knowledge and modeling requirements and ends with the deployment of a usable modeling tool (more details on its phases are available in [4]). This methodology is reduced here to its design and implementation phases, quickly leading to an intuitive and usable result even in the absence of introductory metamodeling theorization. The two iterations are exemplified in this paper, with the initial iteration satisfying the constraint of "not more than 3 concepts".

3.2. Initial Iteration

The initial design phase starts with (i) sketching a mock-up of how diagrams should look in the language being developed and (ii) identifying the distinct types for each element present in the mock-up diagram. The types (node types and connector types) will form the metamodel, introduced here as the "language vocabulary" or "knowledge schema", thus simplifying the traditional notion of meta-modeling established in the MOF specification [19] to one easily compared with the data records-data schema relation.

Figure 2 shows such a diagram mockup depicting a rudimentary process flow (simple sequence of maintenance steps), where each step can be connected either to a sensor or a device it acts upon; sensors should be attachable to devices.

The language vocabulary is introduced as the aggregate answer to four questions: (i) what types of nodes are used in the mock-up? (ii) what types of connectors are used? (iii) what types of nodes should be linked by each connector (i.e., the domain and range of each relation)? (iv) how should the types be unified in order to have a single domain and range for each relation? (i.e., a generalization of the RESOURCE concept is introduced, to allow a maintenance step to act on both SENSORs and DEVICEs).

Non-specialized wording is employed ("types/concepts", "connectors", "generalization", "language vocabulary") to support Business Administration students while at the same time allowing those with computer science background the mapping to a more technical dialect ("classes", "inheritance", "metamodel").

Following this design, students are guided to stepwise implement it on the language engineering component of ADOxx. Implementation phases are clearly distinguished by the building block they address: (i) abstract syntax (the definition of types and their syntactic constraints – i.e., domain, range, cardinality); (ii) notation (the custom graphic symbols attached to each concept and connector); (iii) semantics (the meaning attached to each symbol).

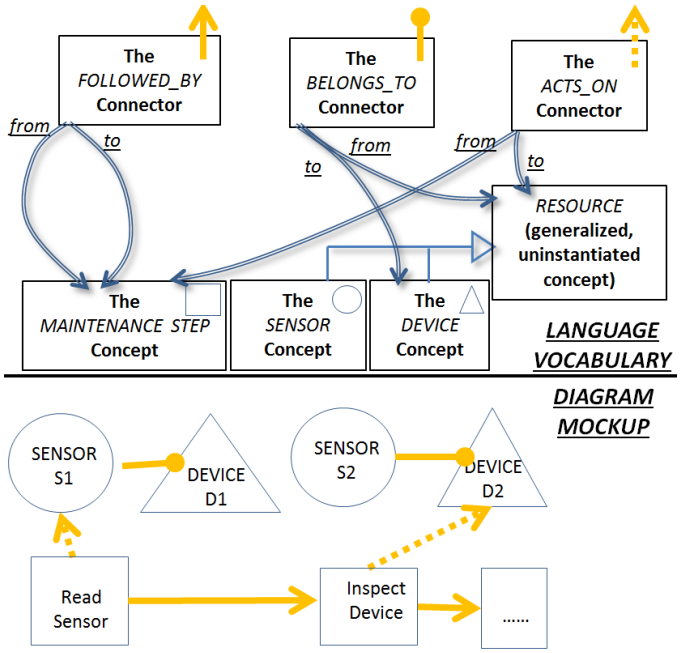

Fig. 2. Diagram mockup and derived language vocabulary

The importance of semantics is stressed as the core benefit of Conceptual Modeling in contrast to free sketching/drawing. Human interpretation and machine interpretation are thus distinguished – the first relying on expressive labeling and visual cues; the second requiring machine-readable (possibly domain-specific) annotation properties that will be exposed to model queries and model-driven systems. These properties must conform a schema that can be tailored for each concept. In this case, to DEVICEs we add a TYPE property (as a way of distinguishing meaning without having to add new graphical symbols to the language) and a DOCUMENTATION property (a hyperlink to some device documentation available outside the modeling tool). Both labels and annotations will later become the basis of running semantic queries against the RDF graph structure that can be derived from models (thus revealing their "knowledge" quality).

Figure 3 shows a model created with the initial modeling tool implementation.

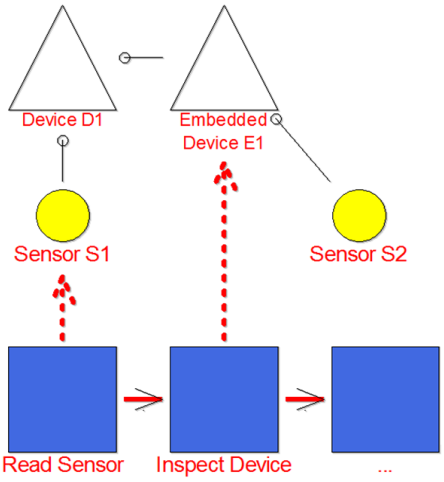

Fig. 3. Model created with the initial language iteration

After the initial language implementation, the other two building blocks of a modeling method are demonstrated: mechanisms and the modeling procedure. A minimal demonstrative mechanism is scripted with the help of ADOxx's internal scripting language. The script shown in Listing 1 captures the event of drawing a connector instance and writes in a log file information about the created connector (which objects have been connected, in what model). It showcases the machine-readable nature of models – through functions

that retrieve the objects and types associated to a modeling event (here, connector creation) while at the same time accessing the external file system to produce output based on model contents.

This is presented as a toy example of traditional model-driven approaches such as code generation. It also introduces the third component of a modeling method, the "modeling procedure" (i.e., the recommended steps for creating models). If the modeling procedure is simple enough to be formalized as a sequence of modeling actions, a "reference sequence" can be compared with logged sequences of various modelers with the help of similarity metrics – e.g., Levenshtein distance. Recent works concerned with the effectiveness of teaching Conceptual Modeling show a growing interest in measuring modeling actions as means of assessing learning outcomes [20] (an approach that we label as "modeling procedure analysis").

```
ON_EVENT "AfterCreateModelingConnector"{
   CC "Modeling" GET_ACT_MODEL
   CC "Core" GET_MODEL_INFO modelid:(modelid)
   CC "Core" GET_CLASS_NAME classid:(classid)
   CC "Core" GET_OBJ_NAME objid:(fromobjid)
   SET sourcename:(objname)
   CC "Core" GET_OBJ_NAME objid:(toobjid)
   SET targetname:(objname)
   CC "AdoScript" FWRITE file:"C:\\log\\log.txt" text:("In model "+modelname+" you
created a connector of type "+classname+ " from object "+sourcename+" to object
"+targetname+"\n") append:yes}
```

Listing 1. ADOxx script for logging modeling actions

3.3. Advanced Iteration

Coming from initial modeling experiences in software engineering, students tend to perceive modeling languages as invariants. However, agility principles, well established in software engineering, may also be adopted for modeling languages/methods. This is demonstrated in our teaching case by evolving the "modeling requirements", followed by a quick reprotoyping of the modeling tool. Examples of requirements driving the new iteration are the following:

a. The maintenance procedure should be more than a sequence of STEPs. DECISIONs may also be necessary;
b. To avoid "construct overload" (cf. [21]), the ACTS_ON relation must be specialized for sensors (READS_VALUE) and devices (ACTS_ON_DEVICE), consequently reducing the processing effort in model queries;
c. To avoid visual cluttering, the modeling language should be partitioned in two distinct types of models (the process and the resources); consequently, the ACTS_ON connector is not only specialized, but also replaced with hyperlinks between models;
d. To improve expressivity, domain-specificity should also be assimilated in notation (as visual cues, plus the freedom to load preferred icons instead of the default symbols);
e. To improve interoperability, domain-specificity should be assimilated in semantics as well (sensors should have a live ADDRESS property that directly gives access to their value stream).

Figure 4 shows diagrams created with the new iteration of the modeling tool.

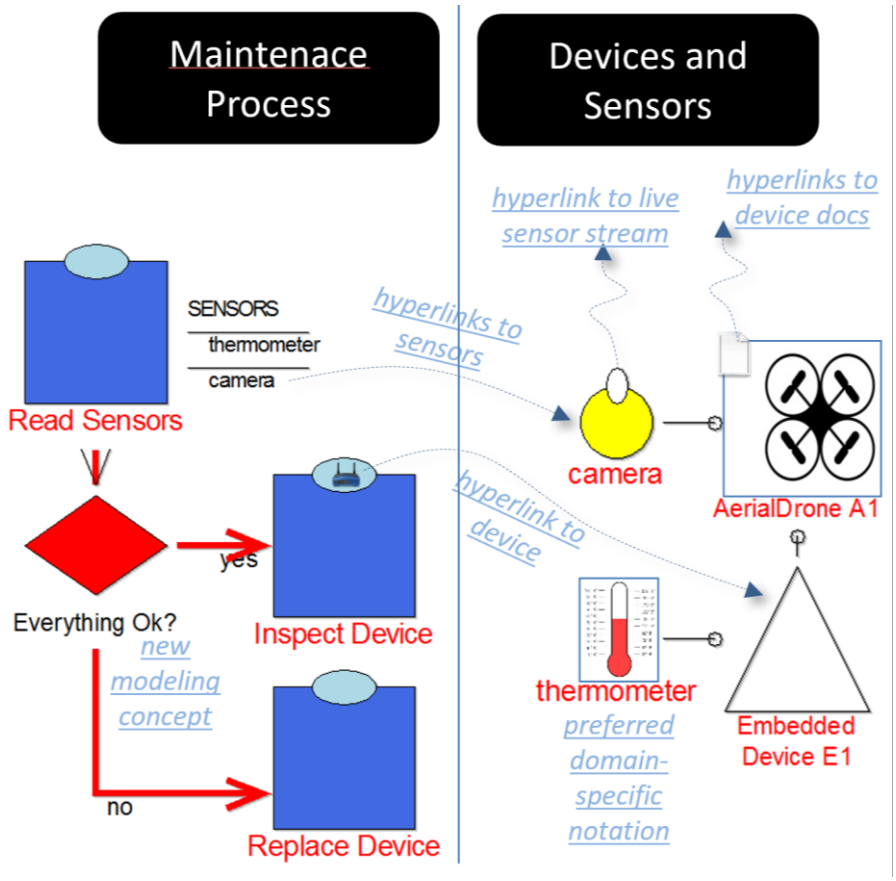

Fig. 4. Models created with the second language iteration

Modeling standards, since they establish consensus, provide a foundation for model compilers and roundtrip engineering. Since in this case we are advocating a non-standard, unpredictable customization of a modeling language, the benefits of consensus do not apply. However, software artifacts can still benefit from the knowledge captured in diagrammatic form - by resorting to the Resource Description Framework.

For this purpose an ADOxx plug-in can convert any type of model (created with any language/tool implemented on ADOxx) to RDF graphs according to certain transformation patterns available in the literature [22]. The derived graphs are hosted by a graph database and some simple model query examples are demonstrated, suggesting the possibility of building client applications that are "aware" of model contents and their "knowledge schema". This specific type of model-driven software engineering method has been discussed in more detail in [23].

An example of a (SPARQL) query is provided here in Listing 2. It retrieves all the devices inspected during a selected procedure and the components attached to them on any decomposition level. Such examples are followed by discussion on the model queries made possible by this advanced iteration – i.e., the new version of the modeling language provides a richer knowledge schema/semantic space, not only visual customizations. At this point RDF and semantic queries are not necessarily mastered by students - this is an introductory example for a subsequent Knowledge Representation module that complements the Conceptual Modeling module by delving into semantic technology.

```
SELECT ?device ?component
WHERE {
GRAPH :MaintenanceProcedure {?x :ActsOnDevice ?device.
                ?device :describedIn ?model}
GRAPH ?model {?component :BelongsTo+ ?device}}
```

Listing 2. SPARQL queries on agile model contents

4. Outcomes

The hands-on demonstration and exercise have proven successful in defusing the fallacies detailed in Section 2 and in establishing a uniform baseline for students regardless of their background. Success is mainly manifested in the sense that more sophisticated model-driven thesis projects have been enabled, moving away from a "blueprint thinking" towards lateral thinking, revealing a more general value and application possibilities for Conceptual Modeling.

Relative to Bloom's taxonomy of educational objectives, we consider that confining students to the role of users (of a modeling language) locks them in limited comprehension, whereas the proposed teaching artifact opens new layers of: *comprehension* ("I understand the role of knowledge schema that a modeling language fulfils, regardless of the application domain"), *analysis* ("I can distinguish the building blocks of a modeling method, I know which is affected by an agile change request"), *synthesis* ("I can synthesize a new method/tool based on my knowledge of those building blocks"), *application* ("I can implement a domain-specific modeling tool"), *evaluation* ("I can relate a modeling method to the requirements/purpose for which it was built").

Regarding a quantified success of the approach, the proposal is part of a master exam that reflects a normal distribution of grading and student interest, just like any other typical exam (about 25% of students propose new domains and bring novelty in their exam projects; 45% do not bother to innovate but are capable of extending the developed prototype with new concepts, attributes, scripts; 30% show little interest or have difficulties passing the exam – this is however not a discrepancy from other exams).

A more relevant outcome than the grading distribution is the fact that our master students were for the first time able to publish scientific works on Conceptual Modeling topics at prestigious international conferences: ENASE 2018 [23], ICEIS 2018 [24] CAISE 2018 workshops [25] and PoEM 2018 workshops [26]. Furthermore the learning curve for junior researchers starting project work was shortened by an estimated 2 months for those attending strictly this 3 meetings module or by 6 months for those following the full program, which includes additional related topics: reflective comparison against well-known modeling languages (e.g., the BEE-UP tool supporting UML, BPMN, ER, EPC and Petri Nets [27]); further reading on the benefits of domain-specific or situational method engineering [11,28]; exercises with graph databases and semantic technology. Of course, this estimation is based on isolated cases as the module was only recently launched and we do not have yet alumni data for a longitudinal survey – besides having alleviated the gap between bachelor studies and design research, we also expect benefits pertaining to jobs related to Business Process Management, Robotic Process Automation and related interoperability scenarios.

5. Related Works

Recent works show growing preoccupation with deploying teaching methodologies for Conceptual Modeling. A recent panel discussion made the following position statement referring to Conceptual Modeling education [29]: "Supportive means such as text books, case examples are hardly available. In many cases teaching may boil down to an art being passed on to students. [...] (basic) courses are dominated by the coding exercise, i.e., students' efforts in mastering simulation software, or, to a lesser degree, statistics associated with model elements or outputs. Hence little time is left for Conceptual Modeling." The work at hand tries to address this discontinuity which, in the authors' experience is often met between bachelor programs and more advanced studies or research project work.

Works such as [20,30] employ analysis of modeling action logs to assess different dimensions of learning and educational objectives when teaching software engineering-oriented Conceptual Modeling (e.g., UML). Another work that quantifies UML model creation errors by novices is [31]. In [32] authors proposed a tool for monitoring the interactions and mistakes done during using an established modeling language. In comparison, our work focuses on empowering students to take control over their modeling method and tool, while being aware of and guided by a purpose and related requirements.

This work was also inspired by previous publications detailing teaching experiences that make use of similar resources (i.e., ADOxx): the modeling tool presented in [33] is much more complex and does not target our specific learning objectives (e.g., minimalism); the case of [34] is closer in scope to our work – however it is subordinated to teaching software engineering (SQL generation from Entity Relationship diagrams). A long-term teaching experience report oriented towards system architect practitioners, rather than students, is presented in [35]. A generalized framework for teaching Conceptual Modeling has recently been published in [36], using a revised variant of Bloom's taxonomy as a motivational starting point. Our future work will provide further analysis of our proposed artifact through the analytical lens established by that publication.

6. Conclusions

The paper introduced a minimalist modeling method as a teaching artifact that can be created together with students, with the help of open use educational resources. Its qualities are minimalism, intuitive constructivism, open-endedness, domain-specificity, detachment from standard practices (while still showing relevance for Software Engineering). The methodological and technological enablers of the proposed artifact are the Agile Modeling Method Engineering framework, the Resource Description Framework and open use available tooling supporting these frameworks.

The framework proposed very recently by [36] will be employed in the next phase of our work to dissect the hereby proposed teaching artifact in terms of the knowledge and cognitive dimensions of the revised Bloom taxonomy.

Future work will also be invested in defining variations of this artifact for other domains and timeframes. Examples of candidate domains are Service Design (for marketing experts) and Narrative Structure Analysis (for communication theorists).

We aim to further reshape this demonstration in order to fit the 2-3 hours frame typically allowed in conference tutorials, as well as to fit it in the curriculum of the Next Generation Enterprise Modeling summer school series [37].

References

1. Embley, D.W., Liddle, S.W. and Lonsdale, D.W.: Principled Pragmatism: A Guide to the Adaptation of Ideas from Philosophical Disciplines to Conceptual Modeling. In: De Troyer, O., Bauzer Medeiros, C., Billen, R., Hallot, P., Simitsis, A., Van Mingroot, H. (eds.) Advances in Conceptual Modeling. Recent Developments and New Directions (ER 2011), Lecture Notes in Computer Science, Vol. 6999, pp. 183-192. Springer, Berlin, Heidelberg (2011)
2. Moser, C., Buchmann, R.A., Utz, W. and Karagiannis, D.: CE-SIB: A Modelling Method Plug-in for Managing Standards. In: Mayr, H. C., Guizzardi, G., Ma H., Pastor O. (eds.) Conceptual Modeling. Enterprise Architectures (ER 2017), Lecture Notes in Computer Science, Vol. 10650, pp. 21-35. Springer, Cham (2017)
3. Strecker, S., Baumol, U., Karagiannis, D., Koschmider, A., Snoeck, M., Zarnekow, R.: Five inspiring course (re-)designs. Business & Information Systems Engineering, 61(2), 241-252 (2019)
4. Karagiannis, D.: Conceptual Modelling Methods: The AMME Agile Engineering Approach. In: Silaghi G. C., Buchmann R. A. and Boja C. (eds.), Informatics in Economy (IE 2016), Lecture Notes in Business Information Processing, Vol. 273, pp. 3-19. Springer, Cham (2018)
5. Karagiannis, D., Mayr, H. C., Mylopoulos, J.: Domain-specific Conceptual Modeling, Springer International Publishing (2016)
6. Karagiannis, D., Kühn, H.: Metamodelling Platforms. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) E-Commerce and Web Technologies. EC-Web 2002, Lecture Notes in Computer Science, Vol. 2455, pp. 182. Springer-Verlag, Berlin, Heidelberg (2002)
7. Prat, N., Comyn-Wattiau, I., Akoka, J.: Artifact Evaluation in Information Systems Design-Science Research-a Holistic View. In: Proceedings of the 19th Pacific Asia Conference on Information Systems (PACIS 2014), AIS (2014)

8. Nonaka, I., von Krogh, G.: Perspective - Tacit Knowledge and Knowledge Conversion: Controversy and Advancement in Organisational Knowledge Creation Theory. Organisation Science 20(3), 635-652 (2009)
9. Karagiannis, D., Buchmann, R., Walch M.: How can diagrammatic conceptual modelling support Knowledge Management? In: Proceedings of the 25th European Conference on Information Systems (ECIS 2017), pp. 1568-1583 (2017)
10. W3C: Resource Description Framework – official website, http://www.w3.org/RDF. Accessed June 24, 2019
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, Heidelberg (2013)
12. Buchmann, R.A., Ghiran A.: Engineering the Cooking Recipe Modelling Method: a Teaching Experience Report. In: CEUR-WS vol. 1999, paper 5 (2017)
13. Bloom, B. S.: Taxonomy of educational objectives: the classification of educational goals: handbook I: cognitive domain, D. McKay (1956)
14. BOC GmbH: ADOxx metamodeling platform – official website, http://www.adoxx.org/live/home. Accessed June 24, 2019
15. Ontotext: GraphDB - official website, http://graphdb.ontotext.com/. Accessed June 24, 2019
16. OMILab: ADOxx-to-RDF download page, http://austria.omilab.org/psm/content/ comvantage/ downloadlist?view=downloads. Accessed June 24, 2019
17. W3C: SPARQL 1.1 Query Language, http://www.w3.org/TR/2013/REC-sparql11-query-20130321. Accessed June 24, 2019
18. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering. Springer-Verlag Berlin Heidelberg (2014)
19. OMG: Meta-Modeling and the OMG Meta Object Facility, https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf. Accessed June 24, 2019
20. Snoeck, M.: Conceptual modelling: How to do it right? In: Proceedings of 11th Research Challenges in Information Science (RCIS 2017). IEEE (2017)
21. Moody, D.L., Heymans, P., Matulevičius, R.: Improving the effectiveness of visual representations in requirements engineering: An evaluation of i* visual syntax. In: Proceedings of 17th Requirements Engineering Conference, (RE 2009), pp. 171-180, IEEE (2009)
22. Karagiannis, D. and Buchmann, R.A.: Linked open models: extending Linked Open Data with conceptual model information. In Information Systems. Vol. 56, 174-197 (2016)
23. Buchmann, R.A., Cinpoeru, M., Harkai, A., Karagiannis, D.: Model-Aware Software Engineering - A Knowledge-based Approach to Model-Driven Software Engineering. In: Damiani, E., Spanoudakis, G., Maciaszek, L. (eds.) Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018), pp. 233 – 240, ScitePress (2018)
24. Harkai, A., Cinpoeru, M., Buchmann, R.A.: Repurposing Zachman Framework Principles for "Enterprise Model"-Driven Engineering. In: Hammoudi, S., Smialek, M., Camp, O., Filipe, J. (eds.) Proceedings of the 20th International Conference on Enterprise Information Systems - Volume 2 (ICEIS 2018), pp. 682-689, ScitePress (2018)
25. Harkai, A., Cinpoeru, M., Buchmann, R.A.: The What Facet of the Zachman Framework: a Linked Data-driven Interpretation. In: Matulevičius, R., Dijkman, R. (eds.) Proceedings of CAISE 2018 Workshops, Lecture Notes in Business Information Processing, vol 316, pp.197-208, Springer (2018)
26. Chis-Ratiu, A., Buchmann, R.A.: Design and Implementation of a Diagrammatic Tool for Creating RDF Graphs. In: CEUR-WS vol. 2238, pp.37-48(2018)
27. OMiLAB: Bee-Up – official website: http://austria.omilab.org/psm/content/bee-up/info. Accessed June 24, 2019
28. Bucher, T., Klesse, M., Kurpjuweit, S. and Winter, R.: Situational Method Engineering. In: Situational Method Engineering: Fundamentals and Experiences. In: Ralyté J., Brinkkemper, S., Henderson-Sellers B. (eds.) IFIP — The International Federation for Information Processing, Vol. 244, pp. 33-48. Springer, Boston, MA (2007)

29. van der Zee, D.J., Kotiadis, K., Tako, A.A., Pidd, M., Balci, O., Tolk, A. and Elder, M.: Panel discussion: education on conceptual modeling for simulation – challenging the art. In: Proceedings of the 2010 Winter Simulation Conference (WSC 2010). pp. 290-304. IEEE (2010)
30. Bogdanova, D., Snoeck, M.: Domain Modelling in Bloom: Deciphering How We Teach It. In: Poels, G., Gailly, F., Serral Asensio, E., Snoeck, M. (eds) The Practice of Enterprise Modeling (PoEM 2017). Lecture Notes in Business Information Processing, vol 305. pp. 3-17. Springer, Cham (2017)
31. Kayama, M., Ogata, S., Masymoto, K., Hashimoto, M., Otani, M.: A practical Conceptual Modeling teaching method based on quantitative error analyses for novices learning to create error-free simple class diagrams. In: Proceedings of 3rd Advanced Applied Informatics (IIAI 2014), pp. 616-622. IEEE (2014)
32. Ternes, B., Strecker, S.: A web-based modeling tool for studying the learning of conceptual modeling, Modellierung 2018, Gesellschaft für Informatik, pp. 325-328 (2018)
33. Bork, D., Buchmann, R., Hawryszkiewycz, I., Karagiannis, D., Tantouris, N.,Walch, M.: Using Conceptual Modeling to support innovation challenges in Smart Cities. In: Proceedings of IEEE 14th International Conference on Smart City (SmartCity 2016). pp. 1317-1324, IEEE (2016)
34. Glässner, T.M., Heumann, F., Keßler L., Härer, F., Steffan, A., Fill, H.G.: Experiences from the Implementation of a Structured-Entity-Relationship Modeling Method in a Student Project. In: Proceedings of the 1st International Workshop on Practicing Open Enterprise Modeling within OMiLAB (PoEM 2017), Vol. 1999 (2017)
35. Muller, G.: Challenges in Teaching Conceptual Modeling for Systems Architecting. In: Jeusfeld M., Karlapalem K. (eds.) Advances in Conceptual Modeling. ER 2015. Lecture Notes in Computer Science, Vol. 9382. pp. 317-326, Springer, Cham (2015)
36. Bork, D.: A Framework for Teaching Conceptual Modeling and Metamodeling Based on Bloom's Revised Taxonomy of Educational Objectives. In: 52nd Annual Hawaii International Conference on System Sciences (HICSS 2019), pp. 7701-7710 (2019)
37. OMILab: NEMO Summer School Series, http://nemo.omilab.org/2018/. Accessed June 24, 2019