Engineering the Cooking Recipe Modelling Method: a Teaching Experience Report

Robert Andrei Buchmann, Ana-Maria Ghiran

Business Informatics Research Centre,
Faculty of Economic Sciences and Business Administration,
Babeş-Bolyai University, Cluj Napoca, Romania
{robert.buchmann,anamaria.ghiran@econ.ubbcluj.ro}

Abstract. A recently introduced master program offering a Business Modelling track in the authors' university includes several courses that touch on conceptual modelling topics. They cover both modelling with well-known languages (UML, BPMN etc.), as well as the development of domain-specific modelling software driven by targeted requirements. Consequently, the study program aims to support two complementary modeller perspectives: (i) of the modellers who adopt "de jure" or "de facto" standards that are reusable across domains; (ii) of the modellers who prefer to have a modelling tool tailored for specific needs and their preferred domain-specific abstraction. While the first perspective is quite common and well served in Business Information Systems curricula, the second perspective requires a well-designed minimal case capable of fostering deep understanding of a modelling method's building blocks and of the methodology needed to tailor or evolve those building blocks towards satisfying explicit requirements. The paper reports on such a teaching experience and provides insights regarding its learning design rationale.

Keywords: Teaching Conceptual Modelling, Domain-specific Modelling, Metamodelling, Agile Modelling Method Engineering

1 Introduction

The goal of the paper is to reflect on the teaching experience and the design rationale of a teaching case developed for a conceptual modelling study track covering both practitioner-oriented and research-oriented topics. The study program was proposed to align the local Business Information Systems curriculum to various study programs around Europe, and to benefit from the teachers' experience and interactions with the community of experts giving lectures in the Next Generation Enterprise Modelling (NEMO) Summer School series [1]. The intent was to compensate for a weak representation of conceptual modelling topics in Romanian university programs, as the majority of investigated programs present such topics as ancillary to courses on *Databases* (e.g., Entity-Relationships Diagrams) or *Software Engineering* courses (e.g., UML). Rather than having conceptual modelling perceived as subordinated to software design concerns, the goal of this track was to position conceptual modelling

under the Design Science paradigm [2], emphasising its specific artefacts, practices and the general value for models as results of knowledge externalisation.

The profile of the master students participating in the hereby discussed teaching experience is summarised as follows:

Student background: The majority of students graduated a Business Information Systems (IS) program or a Computer Science (CS) program. A minority (<10%) graduated a Business Administration (BA) program.

Student experience with modelling: The majority (IS/CS graduates) used UML in MS Visio as means of graphically documenting their bachelor thesis. ER diagrams were used in their Database courses. The BA graduates prefer to use Powerpoint diagramming – typically for organisational charts and process diagrams - due to the lack of constraints ("the freedom of drawing"). The students have a general intuition that not all models are correct, but take a rather intuitive approach in assessing this.

Student understanding of modelling goals: All students agree that the main goal of modelling is to support human understanding of a system design through graphical documentation, as alternative to written text which is "tiring to read". Some CS graduates are aware about the "code generation" use case but have never employed it.

Considering this profile, a showcase modelling method was designed and employed as a teaching artefact, in order to emphasise its engineering process in a tutorial style and to widen the perception on the role and means of conceptual modelling. This experience and its learned lessons are further shared in this paper. The remainder of the paper is structured as follows: Section 2 will decompose this initial state of affairs in more granular teaching challenges and concerns. Section 3 will discuss methodological aspects. Section 4 will overview the modelling method employed as a teaching artefact. Section 5 will summarise findings via a SWOT analysis. Section 6 will comment on related works. The paper ends with conclusions.

2 Teaching Challenges

The initial state of affairs (summarised in the Introduction) is decomposed in Table 1 in several directly targeted "issues" that needed to be alleviated, considering the initial understanding of students about certain key concepts.

3 Methodology

3.1 Teaching Methodology

Overarching the identified teaching challenges, the tutorial's design rationale is also driven by several targeted meta-qualities: (i) *minimalism* to ensure quick comprehension, requiring only trivial domain expertise, (ii) *domain-specificity* manifesting in all building blocks of the engineered method, (iii) an *evolutionary* approach highlighting how each method building block can evolve under an assumption of evolving requirements, (iv) a *constructivist* approach [3] aiming to stimulate lateral thinking and

the generation of understanding by forcing a clash between what is already known (preconceptions about modelling means, goals and purposes) and what is revealed through hands-on experience.

Table 1. Targeted teaching challenges

Issue A: Understanding the goals and value of "modelling at large"
Initial assessment: Limited understanding of modelling goals. All students agree that the goal of modelling is graphical documentation of system design or requirements. Some CS graduates are aware about the "code generation" goal (they can point to the UML Class Diagram, as a potential source for code generation). Inability to provide a simple Knowledge Management (KM) scenario that can use models.
Teaching goal: To shift the student understanding from "modelling is drawing" to "modelling is knowledge representation". To raise awareness on the Knowledge Management paradigm.
Issue B: Understanding the modelling method building blocks
Initial assessment: Weak understanding of distinctions between notation, syntax, semantics, modelling procedure, modelling method, modelling functionality. Students intuitively distinguish between syntax ("pertaining to form") and semantics ("pertaining to meaning").
Teaching goal: To clarify these distinctions.
Issue C: Understanding the semantics of "semantics"
Initial assessment: Confusion regarding the distinction between *human-oriented semantics* and *machine-oriented semantics*. When asked "who is the main consumer of models – humans or machines?" - the majority answers "humans" ("machines do not need diagrams").
Teaching goal: To clarify the different semantic levels of diagrammatic models.
Issue D: Understanding model "correctness"
Initial assessment: Weak understanding of model qualities and model-to-reality relation. When asked "what is a correct model?" the common answer is "a model that accurately represents reality".
Teaching goal: To reveal the axiomatic principle that "all models are wrong, but some are better than others" (according to the model's purpose).
Issue E: Understanding the application domain of conceptual modelling
Initial assessment: Weak understanding of the application domain for conceptual modelling. When asked "where is conceptual modelling applied?" the common answer is "software engineering".
Teaching goal: To emphasise the general value of models as means of knowledge externalisation independent of domain (i.e., adaptable to any application domain).
Issue F: Understanding agility at "modelling method level"
Initial assessment: A recurring idea that all modelling should be done with UML "because it is a standard". General lack of awareness of the Design Science paradigm and of the fact that modelling software/languages/methods are artefacts subjected to design/development processes.
Teaching goal: To shift the student understanding that a modelling language is a given invariant towards the understanding that agility principles from software development can be easily transferred to modelling method implementation. To raise awareness on the Design Science paradigm.
Issue G: Low awareness of conceptualisation activities
Initial assessment: The general perception is that a modelling language is a set of graphical symbols to which meaning is assigned, rather than a conceptualisation whose elements have visual representation. When asked to make a presentation about BPMN, the students' discourse starts from the "numerous graphical symbols" rather than from the set of concepts.
Teaching goal: To create awareness on conceptualisation efforts that stand behind a modelling method.

By aggregating the teaching goals in Table 1, several high-level goals are formulated in Fig. 1(a) – these are the "targeted revelations" to be induced to students with respect to established Information Systems paradigms: (i) establishing the role of a modelling method as a means of externalising "query-able" knowledge (and not only as support for graphical documentation or code generation); (ii) establishing the role of a modelling method as a Design Science artefact that can be tailored for a specific purpose/usefulness and is subjected to a specific engineering process involving both

abstract conceptualisation and implementation of usable tooling; (iii) introducing the "agility" quality of a modelling method; (iv) establishing the non-fixed application scope of modelling methods (complementing the popular use in software design).

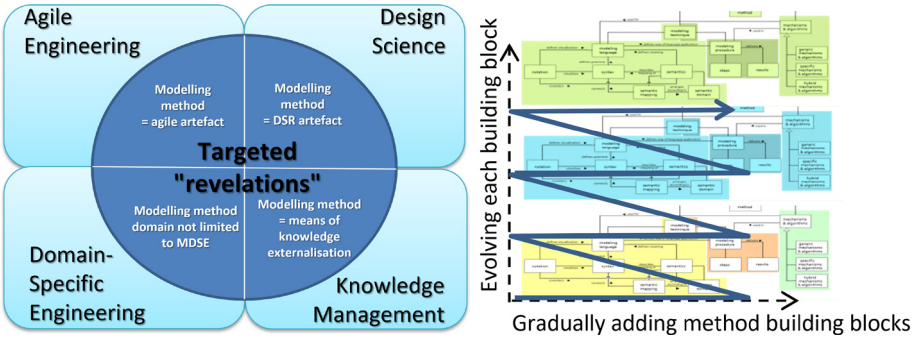

Fig. 1. The paradigms grounded in the proposed teaching case (a) and the orthogonal criteria for incrementing the teaching artefact (b)

Process-wise, the teaching starts by presenting the modelling method building blocks cf. the definition of [4]. Then a sample modelling method is implemented incrementally on the ADOxx platform [5], in small increments that are immediately imitated by students. These increments are accompanied by a theoretical exposé following a bottom-up approach: small building blocks are explained as they are implemented in an additive manner. At the end, students are taken back to the "modelling method" concept, to reflect back on how they worked on its building blocks. Finally, more complex tools are shown as exemplary outcomes to which students can easily relate – e.g., the BEE-UP tool [6-7] allowing modelling with UML, BPMN, EPC, Petri Nets, ER and a few extensions on those languages. When using this tool, students are able to extrapolate from their own implementation experience and, at the same time, they come in contact with a concrete case of applying the Agile Modelling Method Engineering (AMME) framework [8].

3.2 Teaching Artefact Co-Development Methodology

The learning effort is supported by the agile (incremental, iterative and requirements-driven) implementation of a usable artefact – a modelling tool that deploys a domain-specific modelling method designed for knowledge acquisition (i.e., models form a knowledge base which can be queried and processed by machines). Two orthogonal criteria are thus guiding the incremental development, as highlighted in Fig. 1b: (i) gradually adding building blocks to the modelling method; (ii) agilely evolving each building block in relation to satisfy some assumed changing requirements.

The additive and the evolving development are intended to reflect the two manifestations of agility as stated by the AMME framework: (i) *artefact agility* and (ii) *methodological agility*. The first means that a (partial) separation of concerns should be achieved by decomposing the modelling method artefact in constituents – notation,

syntax, semantics, functionality and modelling procedure (derived from the definition in [4]). The latter means that all these constituents can evolve, in order to address changing requirements according to the iterations of the AMME methodology, with each iteration including both conceptualisation efforts and implementation/testing.

4 The Showcase Modelling Method

4.1 The Selected Application Domain

The application domain of *Cooking* was selected for several reasons: (i) to provide a uniform starting point for all students regardless of their background and modelling experience; (ii) to defuse the dominant perception that modelling activities are subordinated to software engineering, as means of graphical documentation for which a "drawing tool" is sufficient; (iii) to show what domain-specificity means without requiring extensive domain expertise; (iv) to be able to make analogies with business process modelling.

The assumed case is that of a Food Establishment whose manager decides to apply a KM strategy of collecting recipes from the hired chefs in the form of a model base that can be later consulted, analysed and transferred to other human resources (cooks, future chefs). The role of modelling in KM was discussed in deep detail in [9]. A Cooking Recipe modelling method and tool are required to externalise knowledge that otherwise would be captured in unstructured, natural language form. The goal of code generation is out of the discussion; the goal of graphical documentation is fulfilled but is not central; the key purpose being the accumulation of a knowledge base that can be processed by machines (queried, at the very least).

4.2 Initial Method Implementation

The **conceptualisation effort** starts with imagining how a diagram describing a cooking recipe should look. Intuitive mock-ups such as the one in Fig. 2 (bottom) are created on blackboard or with Powerpoint, then the types for each diagram element are collected in the "language terminology" layer (top of Fig. 2). The mock-up shows a cooking recipe described as a linear chain of cooking steps, with the Start and Stop clearly distinguished. To enrich the KM value of a recipe (i.e., semantic richness), each step is attached to requirements for an ingredient, a tool and/or further documentation support (e.g., a video showing how to perform the step).

After the types for mock-up elements are collected at metamodel level, the notion of (abstract) **syntax** is introduced and minimally exemplified through *domain and range constraints for visual connectors* – i.e., what types of elements should be connected by each type of connector. Students are already accustomed from natural language grammar (or from programming "syntax errors") that syntax governs the structure of sentences/statements and the order of their constituents. In conceptual modelling, the domain, range (and cardinality, to be introduced later) dictate the order in which elements of different types can be connected in models. Unlike natural language where words are linearly juxtaposed (e.g., left-to-right), in models the elements

are connected in a graph expanding in all 2D directions. Model elements are presented as "words", models as "phrases", and the metamodel as "the language terminology" ("vocabulary", "dictionary"). The analogy with natural language facilitates to bring all students to a common level of understanding without resorting to the specialised jargon of metamodelling (as homework, they are also asked to organise the metamodel as a UML class diagram).

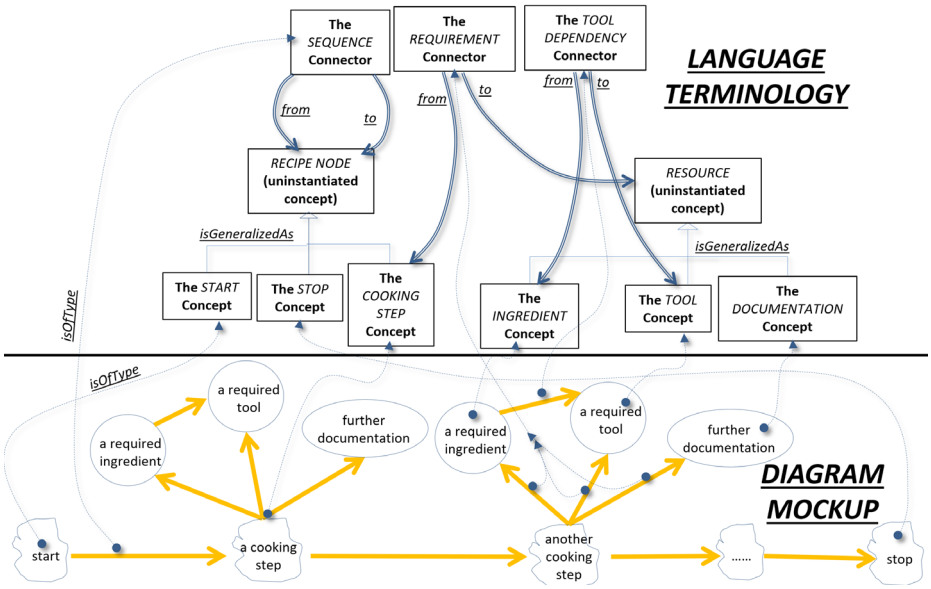

Fig. 2. A diagram mock-up and its governing terminology

Since some relations may connect multiple types of elements, those types are unified under common generalised superclasses presented as "uninstantiated concepts" - i.e., concepts that will not be directly instantiated in models (i.e., no graphical symbol will be devised for them) but are nonetheless useful as a generalisation mechanism. They allow, for example, the SEQUENCE relation to connect elements of *any* of the types START, STOP or COOKING STEP.

The next building block that is introduced is the **notation** (concrete syntax) – i.e., a graphical symbol is attached to each concept and relation. Syntax and notation are implemented in tandem to obtain a usable modelling tool, based on the ADOxx platform [5]. The implemented tool (using minimalistic and quite arbitrarily chosen shapes) allows students to create models such as the one shown in Fig. 3. During implementation, awareness is raised on the need to communicate human-oriented meaning by incorporating labels in graphical symbols, thus establishing a bridge towards the next building block - **semantics**.

Semantics is dominantly understood as the meaning that a user assigns to (or understands from) a model, primarily through labels. However, this is a drastically reductive viewpoint that stands behind the "modelling is drawing" understanding. Even under this understanding, (i) graphical shapes should not be arbitrary shapes (see the

works of Moody on notational qualities [10]); (ii) they should dynamically communicate some changes in meaning (i.e., unlike in pen-and-paper diagramming, a model can communicate properties through notational dynamics and interactivity).

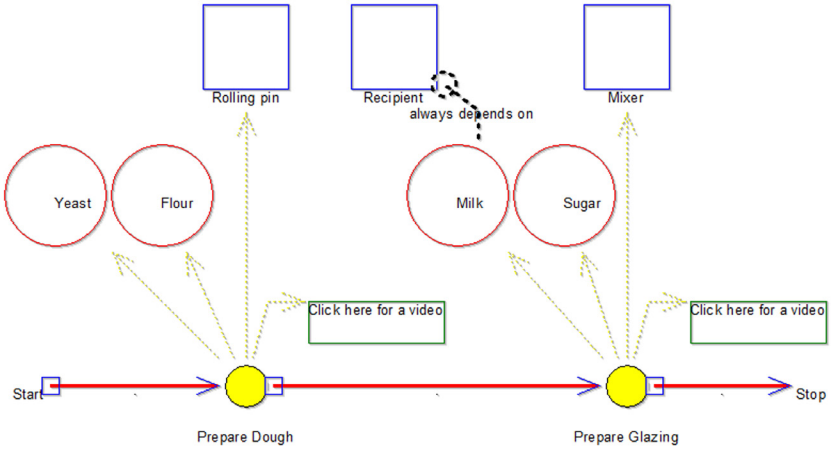

Fig. 3. Model example showcasing the implementation of the initial iteration

As we push towards the "modelling is knowledge representation" perspective, semantics should become machine-readable (i.e., query-able, at least). This is induced via two means: (i) *through syntax*, since the connectivity constraints are dictated by some semantic rationale; (ii) *through conceptional schemata* – i.e., the meaning of elements in the language terminology should rely not only on labels and graphics, but also on machine-readable concept descriptions in the form of properties prescribed for the domain-specific characterisation that we need to capture.

Fig. 4(a) shows an explicit example of such a schema for one of the language terminology concepts (COOKING STEP), thus introducing a clear notion of **domain-specificity** manifesting on semantic level. Based on the machine-readable semantics (induced by either conceptional schema or by syntax), model query **functionality** can be developed. A simple query builder is automatically generated by ADOxx (Fig. 4(b) and Fig. 4(c)) and additional functionality is implementable with the help of its internal programming environment (ADOScripts).

Finally, the last building block of a modelling method – **the procedure** is introduced as a human-oriented (documented or assisted) process of creating models that are adequate for some targeted purpose. This also creates the opportunity of discussing the notion of correctness, emphasising the following:

- the *relativity of model correctness to purpose* – i.e., a process simulation algorithm would require more constraints than the model query functionality, and possibly even more concepts (e.g., decisions) or properties (e.g., probabilities);
- the *variety of means for enforcing correctness*: (i) at interaction level (i.e., when the modellers tries to create an invalid element); (ii) at saving time (i.e., a global check is applied only when saving); (iii) through fully customised functionality

(e.g., checking properties of the conceptional schemata); (iv) through procedural guidance (i.e., leaving the user interaction unconstrained but providing assistance and documentation).

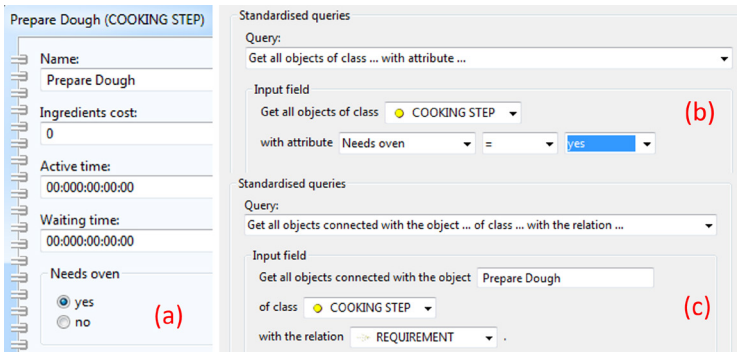

Fig. 4. Conceptional schema of COOKING STEP instantiated for the "Prepare Dough" step (a), model query that uses this schema (b) and model query that uses only the syntax (c)

4.3 Agilely Evolved Method Implementation

The agility quality for modelling methods is introduced in tight relation with that of *modelling requirements* – i.e., requirements that are specifically targeted to the mentioned building blocks. This section only briefly suggests several increments driven by assumed requirements. Table 2 summarises these requirements and the associated increments, while Fig. 5 shows how the evolved version of the modelling language looks after these increments are applied.

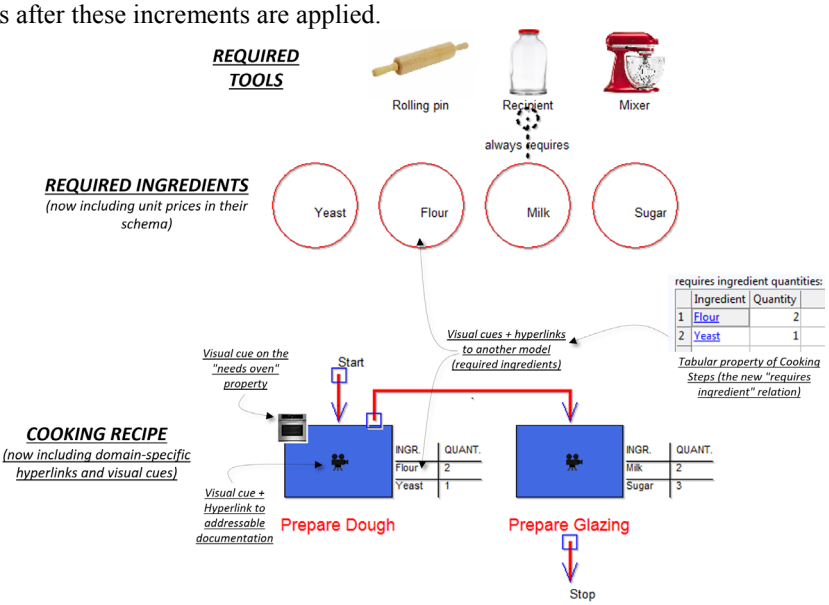

Fig. 5. Model examples according to the evolved version of the modelling language

Table 2. Requirements for evolving the modelling method

Requirement	Solution
"Construct overload" means that semantic distinctions are not mapped on visual distinctions. In the current state of the language, the same relation (REQUIREMENT) is used to connect the ingredient, the tool and the support documentation. Distinction is ensured by the targeted element type, but not by the relation, which is generic for all target types. The disadvantage can be perceived in the model query in Fig. 4c – the query is not be able to distinguish meaning based solely on the connector.	The relations between a cooking step and its ingredients, tools or documentation are differentiated. This distinction may be ensured either by (i) specialising the REQUIREMENT relation for each type of required thing or by (ii) replacing some of its specialisations with hyperlink (see next row).
There's a risk of visual cluttering even in simple models like Fig. 3, due to the numerous REQUIREMENT connectors.	A separation of concerns can be obtained by splitting the modelling language in two types of models: one that describes strictly the recipe (order of steps) and one that catalogues the resources required for recipes. Consequently, the REQUIREMENT visual connector will be replaced by hyperlinks that communicate relations on an interaction level (and through visual anchors rather than connectors).
The DOCUMENTATION concept was created to offer concrete documents (videos, Web pages, PDFs etc.) that describe how to perform a certain cooking step. Therefore it makes more sense to use addressable documents directly linked to model elements.	The DOCUMENTATION concept will be completely removed from the language terminology. Instead, a dedicated hyperlink will be added to the COOKING STEP concept.
Functionality to compute total ingredient cost is needed, based on prices and quantities required along a cooking recipe model.	Prices may be added to the conceptional schema of INGREDIENT, but quantity is not a property of ingredients, it is a property of the relation between COOKING STEP and INGREDIENT. Since we decided above that this relation will not be a visual connector anymore, but a hyperlink, a tabular property is necessary to attach attributes to the hyperlink. Once the new properties are made available and filled with values, a menu item will be added to perform the required calculation with the help of the internal scripting language of ADOxx (and the procedure will be updated to teach users how to use it).
Domain-specificity, which currently manifests on a semantic level, should also be reflected in notation.	The modeller should have the freedom of selecting preferred icons for the required tools or ingredients. Domain-specific information, based on the properties in the conceptional schema, should be communicated through visual cues and hyperlinks that are interactive directly on the elements' notation.

5 Findings

Table 3 reflects on the teaching goals formulated in Table 1 and indicates the explicit means of pursuing them, directly illustrated and commented upon during the students' hands-on experience discussed in the previous sections. In the following, a SWOT analysis summarises the reported experience and its learned lessons:

Strengths: The proposed modelling method has the following teaching qualities: (i) minimalism and ease of implementation (reduced repetitive tasks); (ii) it reveals the notion of modelling method as a Design Science artefact; (iii) it is detached from software engineering and is domain-specific without requiring prior domain expertise; (iv) it relies on free tooling towards deploying a usable result in which students can relate their modelling effort to the design decisions on which it is based.

Weaknesses: When presenting their own homework projects, all students reported process-centric methods. This is due to the behavioural focus of popular modelling

languages (BPMN, Petri Nets, EPC etc.), which is also central to the showcase method example. A false impression was created, that any modelling method should depict behaviour. It is therefore necessary to complement the showcase method with others that share its meta-qualities while avoiding behaviour modelling (i.e., limited to rules-centric or dependency-centric model types) in order to further push the lateral thinking that is stimulated through this teaching case.

Table 3. Specific means of addressing the teaching challenges

Issue A: Understanding the goals and value of "modelling at large"
Approach: Emphasising model queries in contrast with "models as graphics", as means of granularly retrieving the knowledge externalised in model form. Emphasising the dynamic user experience of modelling, as distinguished from pen-and-paper static diagramming.
Issue B: Understanding the modelling method building blocks
Approach: Instilling a gradual "revelation by example" approach, as each building block is introduced separately and evolved additively.
Issue C: Understanding the semantics of "semantics"
Approach: Distinguishing explicitly between human interpretation (label-based or graphics-based) and machine interpretation (based on conceptional schemata and their editable properties to support model queries)
Issue D: Understanding "model correctness"
Approach: Revealing a multitude of means for ensuring correctness relative to the purpose of models: (i) metamodel constraints vs. programmatic constraints, (ii) UI event-based checking (during modelling actions) vs. global model checking, (iii) syntactic constraints vs. semantic constraints.
Issue E: Understanding the application domain of conceptual modelling
Approach: Targeting an application domain far detached from software engineering concerns (and revealing that software engineering is itself an application domain).
Issue F: Understanding agility at "modelling method level"
Approach: Taking an iterative and incremental approach to deploying a modelling method while emphasising its status as a requirements-oriented Design Science artefact.
Issue G: Low awareness on conceptualisation activities
Approach: Creating awareness on the conceptualisation phases leading to the modelling software implementation.

Opportunities: By decoupling conceptual modelling from software engineering, students are stimulated towards lateral thinking and the ability to devise and use modelling methods for domain-specific goals (e.g., business model evaluation, service/value designs). They can transfer their expertise and learned lessons to arbitrary other Business Administration domains (accounting, marketing, management), to develop modelling methods for student projects or theses that are not necessarily subordinated to business informatics.

Threats: Preconceptions and a narrow end-user perspective pose default resistance against the notion of modelling method agility and domain-specificity. A particular threat is raised by how conceptual modelling is introduced to students as subordinated to software development goals and processes. Common practices around the local industry also show (i) a limited goal of models as graphical documentation (lack of awareness on the notions of user-centric modelling goals and modelling requirements); (ii) limited practice and understanding of modelling methods (e.g., Powerpoint drawings with UML shapes rather than UML modelling per se); (iii) lack of awareness on conceptualisation efforts and design rationale underlying any modelling method.

6 Background and Related Works

As the discussed teaching goals indicate, the work at hand contributes to devising effective means for teaching conceptual modelling as a design paradigm for knowledge capture rather than as a practice subordinated to popular use cases or application domains (e.g., software engineering).

The challenge of "how can conceptual modelling education be improved?" is become more relevant in recent years – see the position statements in the panel discussion summarised by [11]: "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 Modelling. Some people may even say little time is "required" as case examples often boil down to close reading rather than modelling."

The proposed teaching case aims to stimulate design thinking as a necessity for any conceptual modelling effort, while also fostering creativity in the sense discussed by [12] in the context of wicked systems. The showcase method was created after studying or acquiring experience with previous teaching cases relying on ADOxx implementations [13-14]. The previous experience was hereby refined to reveal aspects pertaining to (i) Knowledge Management; (ii) Design Science; (iii) agility of method building blocks; (iv) domain-specific modelling requirements.

The authors of [15] also share a masters-level teaching experience regarding conceptual modelling for a narrow domain – however, their discourse relies on quantified (mathematical) means. The author of [16] focuses on assessing, based on logs of modelling events, how students perform modelling tasks with well-known methods. Our focus is rather on giving a complete control over the modelling method, so that students can shift their perspective from that of an end user towards understanding the effort of conceptualising modelling means and constraints, while also reflecting on the design rationale.

7 Conclusions

The paper reported on the teaching experience and design rationale aimed to establish a strong foundation for Business Information Systems master students in understanding the modelling method concept from multiple perspectives – as a Design Science artefact; as means of knowledge externalisation; as an evolvable artefact, adaptable to any domain-specificity; as having a usable manifestation in the form of modelling software. A showcase modelling method for Cooking Recipes was analysed through the lens of the Agile Modelling Method Engineering framework. The teaching goals and means to achieve those goals were analysed in a granular way, relative to the starting state of affairs (initial student understanding and profile). Future work will refine the discussed artefact towards non-behavioural types of models in order to further generalise the learned lessons.

Acknowledgment: The work presented in this paper is supported by the Romanian National Research Authority through UEFISCDI, under grant agreement PN-III-P2-2.1-PED-2016-1140.

References

1. NEMO Summer School Series, http://nemo.omilab.org/2017/
2. Hevner, A. R., March, S. T., Park, J., Ram, S.: Design Science in Information Systems Research, MIS quarterly 28(1):75–105 (2004).
3. Richardson, V.: Constructivist Teaching and Teacher Education: Theory and Practice. In Richardson, V. (ed.): Constructivist Teacher Education: Building a World of New Understandings, pp. 3–14. Routledge (1997)
4. Karagiannis, D., Kühn, H.: Metamodeling Platforms, In: Bauknecht, K., Min Tjoa, A., Quirchmayr, G. (eds.), Proceedings of EC-Web 2002 – DEXA 2002, Aix-en-Provence, France, LNCS 2455, p. 182, Springer (2002)
5. BOC GmbH, The ADOxx metamodelling platform – reference webpage. http://www.adoxx.org/live.
6. Karagiannis, D., Buchmann, R. A., Burzynski, P., Reimer, U., Walch, M.: Fundamental Conceptual Modeling Languages in OMiLAB. In:Karagiannis, D., Mayr, H. C., Mylopoulos, J. (eds.) Domain-specific Conceptual Modelling, pp. 3-30, Springer (2016)
7. OMiLAB, Bee-Up download page, http://austria.omilab.org/psm/content/bee-up/info
8. Karagiannis, D.: Agile Modeling Method Engineering. In: Karanikolas, N., Akoumianakis, D., Mara, N., Vergados, D., Xenos, M. (eds.) Proceedings of PCI 2015, pp. 5-10, ACM (2015)
9. Karagiannis, D., Buchmann, R. A., Walch, M.: How can diagrammatic conceptual modelling support knowledge management? In: Proceedings of ECIS 2017, paper 101, AIS (2017)
10. Moody, D.: The physics of notations: towards a scientific basis for constructing visual notations in software engineering. In: IEEE Transactions on Software Engineering, 35 (5): 756-777 (2009)
11. van der Zee, D. J., Kotiadis, K., Tako, A.A., Pidd, M., Balci, O., Tolk, A., Elder, M., Panel discussion: education on conceptual modeling for simulation – challenging the art, In: Proceedings of the 2010 Winter Simulation Conference, IEEE, pp. 290-304 (2010)
12. Hawryszkiewycz, I., Pradhan, S., Agarwal, R.: Design Thinking as a Framework for Fostering Creativity in Management and Information Systems Teaching Programs. In: The 19th Pacific Asia Conference on Information Systems (PACIS 2015), pp. 5–9, AIS (2015)
13. Bork, D., Fill, H. G., Karagiannis, D., Miron, E. T., Tantouris, N., Walch, M., Conceptual Modelling for Smart Cities: a Teaching Case, Interaction Design & Architectures 27(Winter):10-27 (2015)
14. 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 Int. Conf. On Smart City, IEEE, pp. 1317-1324 (2016)
15. Muller, G.: Teaching conceptual modeling at multiple system levels using multiple views, Procedia CIRP 21, Elsevier, pp. 58-63 (2014)
16. Snoeck, M.: Conceptual modelling: how to do it right? Tutorial presented at the 11th Int. Conf. on Research Challenges in Information Science (RCIS 2017), IEEE, pp. 19 (2017)