Lecture Notes in Information Systems and Organisation 39

Alena Siarheyeva · Chris Barry · Michael Lang · Henry Linger · Christoph Schneider *Editors*

Advances in Information Systems Development

Information Systems Beyond 2020

Springer

Lecture Notes in Information Systems and Organisation

Volume 39

Lecture Notes in Information Systems and Organization—LNISO—is a series of scientific books that explore the current scenario of information systems, in particular IS and organization. The focus on the relationship between IT, IS and organization is the common thread of this collection, which aspires to provide scholars across the world with a point of reference and comparison in the study and research of information systems and organization. LNISO is the publication forum for the community of scholars investigating behavioral and design aspects of IS and organization. The series offers an integrated publication platform for high-quality conferences, symposia and workshops in this field. Materials are published upon a strictly controlled double blind peer review evaluation made by selected reviewers.

LNISO is abstracted/indexed in Scopus

More information about this series at http://www.springer.com/series/11237

Alena Siarheyeva · Chris Barry ·
Michael Lang · Henry Linger ·
Christoph Schneider
Editors

Advances in Information Systems Development

Information Systems Beyond 2020

Springer

Editors
Alena Siarheyeva
130 impasse Louis Bonamici
Résidence le Gouverneur B2
Toulon, France

Chris Barry
Cairnes School of Business and Economics
National University of Ireland Galway
Galway, Ireland

Michael Lang
Cairnes School of Business and Economics
National University of Ireland Galway
Galway, Ireland

Henry Linger
Faculty of Information Technology
Monash University
Melbourne, VIC, Australia

Christoph Schneider
IESE Business School
University of Navarra
Barcelona, Spain

ISSN 2195-4968 ISSN 2195-4976 (electronic)
Lecture Notes in Information Systems and Organisation
ISBN 978-3-030-49643-2 ISBN 978-3-030-49644-9 (eBook)
https://doi.org/10.1007/978-3-030-49644-9

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The **International Conference on Information Systems Development** (ISD) is an academic conference where researchers and practitioners share their knowledge and expertise in the field of information systems (IS) development. As an affiliated conference of the Association for Information Systems (AIS), the ISD conference complements the international network of general IS conferences (ICIS, ECIS, AMCIS, PACIS, HICSS). The ISD conference continues the tradition started with the first Polish-Scandinavian Seminar on Current Trends in Information Systems Development Methodologies, held in Gdansk, Poland, in 1988. This seminar has evolved into the International Conference on Information Systems Development.

Throughout its history, the conference has focused on different aspects, ranging from methodological, infrastructural, and educational challenges in the ISD field to bridging the gaps between industry, academia, and society. Advancements in information systems foster technological developments. The deployment of the resulting technologies in all areas of society, including the public and private sectors, the community, and people's homes is greatly beneficial. ISD has always promoted a close interaction between theory and practice that has set a human-centered agenda focused on advancing the methods, tools, and management of IS development.

This volume is a selection of papers from ISD2019, the 28th Information Systems Development Conference hosted by the Higher Institute for Electronics and Digital Training (ISEN Yncréa Méditerranée) and held in Toulon, France, from August 28–30, 2019. All accepted papers have been published in the AIS eLibrary, which is accessible at https://aisel.aisnet.org/isd2014/proceedings2019. This volume contains extended versions of the best papers, as selected by the ISD2019 Proceedings Editors.

The theme of the conference was *Information Systems Beyond 2020*. It focused on the latest developments in ISD and particularly on emerging concepts, novel approaches, and ideas that are likely to shape information systems research in the 2020s. The conference provided a forum for discussing research and developments in this field.

The ISD2019 conference attracted contributions in the general area of information systems development, as well as in more specialized topics including *Society, Trust, and Ethics in ISD*, *New Media in ISD*, *ISD Methodologies*, *ISD Education*, and *Managing ISD*. ISD2019 focused on these and associated topics in order to promote research into theoretical and methodological issues and ways in which these advances enable better synergies between theory and practice.

We believe that the innovative papers assembled in these lecture notes will inform the reader of important contributions in this regard.

Alena Siarheyeva
Chris Barry
Michael Lang
Henry Linger
Christoph Schneider

Conference Organization

Conference Chair

Alena Siarheyeva

International Steering Committee

Chris Barry
Michael Lang
Henry Linger
Christoph Schneider

Track Chairs

Society, Trust, and Ethics in ISD

Chris Barry
Michael Lang

Exploring New Media in ISD

Jean-Rémy Chardonnet
Ruding Lou
José Tiberio Hernández
Manolya Kavakli

Information Systems Methodologies and Education

Mikko Rajanen
Dorina Rajanen
Monica Vladoiu

Managing ISD

Miguel Mira da Silva
Emilio Insfran
Ana Cristina Ramada Paiva

Current Topics in ISD

Karlheinz Kautz
Sabine Madsen
Sharon Coyle

Reviewers

Jon Aaen
Parisa Aasi
Silvia Abrahao
Ademar Aguiar
Muhammad Ovais Ahmad
Jan Aidemark
Asif Akram
Rafael Almeida
Vasco Amaral
Bo Andersson
Elina Annanperä
Joao Araujo
Leena Arhippainen
Doris Aschenbrenner
Rogerio Atem De Carvalho
Cheuk Hang Au
Eduard Babkin
Per Backlund
João Barata
Chris Barry
Peter Bellström
Olivia Benfeldt Nielsen
Peter Bernus
Gro Bjerknes
Dominique Blouin
Keld Bødker
Veera Boonjing
Ross Brown
Jeremy Brown
Noel Carroll
Priscila Cedillo
Luca Cernuzzi
Narayan Ranjan Chakraborty
Jean-Rémy Chardonnet
Witold Chmielarz
Cesar Collazos
Zoran Constantinescu
Sharon Coyle
Daniela Danciulescu
Duong Dang
Duy Dang Pham
Denis Dennehy
Sean Duignan
João Faria
Marta Fernández-Diego
Jennifer Ferreira
Justin Filippou
Owen Foley
Martin Gellerstedt
Ahmad Ghazawneh
Abel Gómez
Fernando González
Daniel Peter Gozman
Carmine Gravino
Patrick Guillemin
Figen Gul
Darek Haftor
Eija Halkola
Anne Vorre Hansen
Heidi Hartikainen
Henrik Hedberg
Seamus Hill
Mairéad Hogan
Netta Iivari
Yavuz İnal
Emilio Insfran
Amin Jalali
William Jobe
Björn Johansson
Gustaf Juell-Skielse
Miranda Kajtazi
Dimitris Karagiannis
Pasi Karppinen
Karlheinz Kautz
Manolya Kavakli
Rónán Kennedy
Marianne Kinnula
Dina Koutsikouri
Erdelina Kurti
Markus Lahtinen
Arto Lanamäki
Michael Lane
Michael Lang
Birger Lantow
Jouni Lappalainen
Michael Le Duc
J Ola Lindberg
Henry Linger
Ruding Lou
Ulrika Lundh Snis
Sabine Madsen
Monika Magnusson
Tim A. Majchrzak
Jabier Martinez
Raimundas Matulevicius
Frederic Merienne
Miguel Mira Da Silva
Tonja Molin-Juustila
Phelim Murnion
Priyadharshini Muthukannan
Makoto Nakayama
Andrés Adolfo Navarro Newball
Lene Nielsen
Ovidiu Noran
Behnaz Norouzi
Lena-Maria Öberg

Mairead O'Connor
Thuy Duong Oesterreich
Raphael Pereira De Oliveira
Christian Ostlund
Ana Paiva
Malgorzata Pankowska
Nearchos Paspallis
Ruben Pereira
Guillaume Perocheau
John Persson
Daranee Pimchangthong
Tomas Pitner
Dijana Plantak Vukovac
Claudia Pons
Natallia Pshkevich
Iman Raeesi Vanani
Claudia Raibulet
Mikko Rajanen
Dorina Rajanen
Marios Raspopoulos
João Reis
António Rito Silva
Alberto Rodrigues Da Silva
Christoph Rosenkranz
Bruce Rowlands
Paulo Rupino Da Cunha
Mikko Salminen
Kurt Sandkuhl
Sanem Sariel
Wilson Javier Sarmiento
Boubker Sbihi
Helana Scheepers
Christoph Schneider
Ulf Seigerroth
Alena Siarheyeva
Ashok Sivaji
William W. Song
Michiel Spape
Zlatko Stapić
Karen Stendal
Stefan Stieglitz
Janis Stirna
Frantisek Sudzina
Ann Svensson
Torben Tambo
Barney Tan
Kimmo Tarkkanen
José Tiberio Hernández
Justas Trinkunas
Fanny Vainionpää
Juan Manuel Vara
Tero Vartiainen
Christopher Vendome
Monica Vladoiu
Liisa Von Hellens
Neven Vrcek
Ulrika H. Westergren
Anna Wingkvist
Karen Young
Bo Yu
Alfred Zimmermann
Miguel Angel Zúñiga Prieto

Contents

Advancing Conceptual Modeling Education Towards a Generalized Model Value Proposition

Ana-Maria Ghiran, Cristina-Claudia Osman, and Robert Andrei Buchmann(✉)

Business Informatics Research Center, Babeș-Bolyai University, Cluj-Napoca, Romania
{anamaria.ghiran,cristina.osman,robert.buchmann}@econ.ubbcluj.ro

Abstract. This paper proposes a teaching method and artifact for Conceptual Modeling education, motivated by a challenge in the authors' university of bridging the gap between bachelor-level studies and research work on topics related to Conceptual Modeling. At bachelor-level, Conceptual Modeling is subordinated to Software Engineering or Business Process Management topics, making extensive use of available standards for graphical documentation purposes. However, at doctoral level and in project-based work, modeling methods must be scientifically framed within wider-scoped paradigms – e.g. Knowledge Management, Enterprise Modeling – or tailored for domain-specific scenarios. The teaching artifact presented in this paper is an example of an "agile modeling method" that can be iteratively evolved together with students through a metamodeling approach in support of a course flow that argues for a generalized model value proposition and modeling languages acting as "schema" that can be tailored and migrated to accommodate explicit requirements from any application domain.

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

1 Introduction

This paper extends a proposal on improving Conceptual Modeling education presented at ISD 2019 [1], by enriching the proposed teaching artifact with details regarding the course flow that frames it, the rationale of its design decisions and further justification on the choice of tooling and methodology.

Conceptual Modeling education can be tackled as a design problem to address common preconceptions identified in the students' understanding, and possibly in the stance of educators who strictly employ Conceptual Modeling for common use cases (e.g. database design, business process modeling). The hereby advocated teaching approach introduces Conceptual Modeling to students as a purposeful activity that has a value proposition for a diversity of domains, among which databases design or control flow modeling are only a selection of (popular) use cases. Others, such as service design [2],

A prior version of this paper has been published in the ISD2019 Proceedings (http://aisel.aisnet.org/isd2014/proceedings2019).

A. Siarheyeva et al. (Eds.): ISD 2019, LNISO 39, pp. 1–18, 2020.
https://doi.org/10.1007/978-3-030-49644-9_1

smart city simulations [3], value analysis [4] or even a knowledge management approach to something as trivial as cooking recipes [5] can equally benefit from this discipline, as it provides means for capturing structured conceptualizations on a mitigation layer between human-oriented knowledge representation and machine-readable knowledge representation. When we add the ingredient of *agile metamodeling*, modeling languages become schemata that enable domain-specific knowledge retrieval – something that can be convincingly presented by analogy with how a database schema enables SQL queries for data-driven system or decision.

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 of courses on software engineering or business process management. On the other hand, a wider and deeper understanding is required in research work – i.e., project-based industry collaborations, doctoral and postdoctoral studies, some master dissertations (especially in industry collaborations where domain-specificity tends to be a key requirement). In master or doctoral studies it is not sufficient to wear the hat of a modeling tool "user" (who takes a modeling language for granted), but it is often necessary to be capable of exercising abstraction abilities, of expanding standards, of hybridizing modeling dialects or developing model-based proof-of-concept experimentation and evaluation.

This gap in perception is comparable to the one between "database users" (who operate on data records, while taking a 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 which is dispersed in "aspects" of other disciplines. This turns Conceptual Modeling education into a "design problem" (in the sense of Design Research) – a problem we are investigating along the full engineering cycle, from contextual requirements to proposed treatments. As a possible treatment to this problem, we hereby present a teaching artifact that aims to stimulate students' lateral thinking – the main learning objective is to show that a modeling language is a *knowledge schema* to be tailored and migrated in the same sense as a database schema, in order to ensure the semantic richness necessary for some selected purpose (which may go well beyond graphical documentation, e.g. interoperability with external systems). Software Engineering is thus repositioned as an application domain that benefits from standards, best practices and consensus; but at the same time, a more general notion of "model value" is introduced - one that transcends application domains and follows learning design recommendations from the literature [6].

The proposed teaching artifact is a "modeling method" (cf. the definition of [7]) that showcases to students, through a minimalist approach: (i) a modeling method's building blocks, prototyped in the form of a domain-specific modeling tool; and (ii) a conceptualization and implementation process based on the Agile Modeling Method Engineering framework [8] to enable the agile migration of a modeling prototype assuming evolving requirements. Therefore, the Design Research challenge for which this artifact was developed is *How can we teach Conceptual Modeling in a way that stimulates productivity and creativity of students in research work, expanding their understanding of modeling purpose and model value beyond traditional application areas and use cases?*

The short answer, for which the proposed artifact was developed, is *by revealing the "agile schema" function that a modeling language can fulfil and by demonstrating its evolvability driven by requirements.* The targeted success measure in our university's case was to enable master students to publish scientific contributions for the first time in international venues on Conceptual Modeling topics, derived from their own dissertations, thus easing their learning curve towards project-based work and potential doctoral studies.

The remainder of the paper is organized as follows: Sect. 2 clarifies the working terminology and provides justification for the choice of tools. Section 3 outlines the requirements for the proposed teaching artifact and provides an overview on the proposed solution. Section 4 presents the teaching artifact and how it fits in the overall teaching method and course flow. Section 5 discusses observed outcomes. Section 6 comments on related works. The paper ends with conclusions.

2 Working Terminology and Justification

In this work's interpretation, the term *Conceptual Modeling* refers to a "standalone discipline that uses or creates conceptualizations for any domain" [9], resulting in diagrammatic abstractions relevant to that domain and driven by some requirements. The longstanding conference series on Conceptual Modeling (ER), although often presenting Software Engineering use cases, generally recognizes this wide scope - covering from philosophical foundations [10] to expanding application areas, e.g. Enterprise Architecture Management [11]. However, in education (and educational research) this wide scope is obscured; consequently, junior researchers face a steep learning curve when they suddenly discover that modeling is not limited to ancillary techniques they routinely employ for documenting their work in other disciplines.

This paper is motivated by direct observation on the study programs where authors are involved, which is backed by a recent literature survey [12] showing that, perhaps due to how the ACM/AIS curriculum on Information Systems [13] is designed, the research literature on Conceptual Modeling education is dominated by Software Engineering scenarios – e.g. [14, 15], with a minority (quarter) of surveyed works pertaining to Business Process Management and only isolated works tackling other application areas (e.g. [16]) - although the diversity of available modeling languages is otherwise well represented outside educational contexts (from work on extending standards like Archimate [4] to domain-specific projects [17]). Some academic courses where Conceptual Modeling is positioned as a standalone discipline can also be identified [18, 19] but these are tightly coupled to Software Engineering contexts.

The engineering process for this artifact (and the associated tutorial flow) is a simplification of a metamodeling approach called Agile Modeling Method Engineering (AMME) [8], which employs notions of "model", "instance" and "metamodel" similar to the Meta-Object Facility [20] but is independent of the UML language family and aims to support a full production line of modeling tools. Its formal foundation is the FDMM formalism published in [21] aligned with the meta-metamodel of the ADOxx platform [22] that assumes a graph-like underlying structure for any diagrammatic representation. This is also the metamodeling platform employed to develop the artifact together with

students, due to its (i) free availability, (ii) rapid prototyping features that help students produce something usable before (or in parallel with) acquiring theoretical foundations or programming experience, (iii) open access to a diversity of modeling tools that students can dissect and repurpose, hosted within the Open Models Laboratory ecosystem [23]. The design decisions to be presented in this paper aim for educational qualities, a minimization of prerequisite skills and of domain expertise, therefore they should be easy to translate to other preferred platforms, with only few limitations (e.g. built-in model interoperability features are quite diverse among metamodeling platforms and may require some implementation effort on the educator's part).

3 Requirements for the Teaching Artifact

Several meta-requirements have been distilled as motivation for the proposed teaching artifact. These are synthesized in Table 1 together with their rationale, paralleled by suggestions on how they are addressed *(Solution Approach)*.

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" [7] (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 [24])	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 [25] through the lens of Conceptual Modeling (e.g., [26]), modeling is presented as a means of Knowledge Externalization. The "knowledge representation" quality of models is stressed by showcasing the ability of applying semantic queries on models, employing the Resource Description Framework (RDF) [27] 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 [17, 28], 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 knowledge retrieval goals (model queries and possible interoperability with model-driven systems)

(*continued*)

Table 1. (*continued*)

Requirement	Solution approach	Rationale
D. *To reveal the agility potential of modeling methods.*	The Agile Modeling Method Engineering [8] 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

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) – employed strictly as graphical documentation for bachelor theses (typically using drawing tools with diagramming "templates").

The value of models *as purposeful knowledge representation* is thus lost or diluted by the common use case of graphical 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 semantically 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.

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 [5]:

- **Minimalism:** The development of the modeling method should be demonstrable in 2 meetings × 3 h each, plus an additional meeting for discussion (to map the hands-on experience on theoretical background provided by parallel lectures, also suggesting potential extensions for student homework). The modeling language should introduce in its first iteration not more than 3 concepts (and necessary relations), thus reducing the complexity to a "Hello world" kind of demonstration – however one that *touches all building blocks of a modeling method 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 [29] - 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 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 (with 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 therefore a minimalist modeling method – sufficiently rich to showcase the core principles of Agile Modeling Method Engineering and, at the same time, open-ended for further extensions in student homework. The building blocks of this artifact are shown in Fig. 1, each mapped to their enabling technologies (free versions for educational purposes are available for all of these):

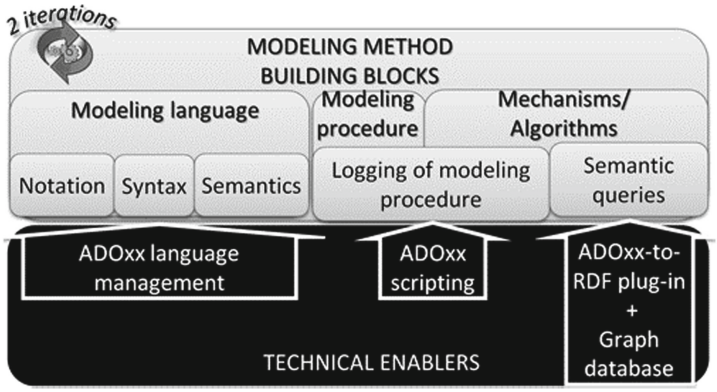

Fig. 1. Building blocks and enablers of the proposed teaching artifact

1. ADOxx [22], a metamodeling platform for rapid prototyping of modeling tools (including notation, syntactic rules, semantics or model-driven functionality);
2. ADOScript, the built-in scripting language of ADOxx for implementing model-based functionality;
3. GraphDB [30], an RDF graph database server with ontological capabilities (to store models as knowledge graphs and demonstrate model queries outside the modeling environment);
4. ADOxx-to-RDF [31], a plug-in for converting diagrammatic models to 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 [32]) from arbitrary clients.

4 Methodology and Artifact

4.1 Application Domain for the Teaching Artifact

The research method underlying this work is subordinated to the Design Science research paradigm [33] – 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 knowledge creators guided by specific requirements in a narrow application domain where a modeling layer must be employed to bridge human understanding and a technological execution environment. Thus the artifact is iteratively built to defuse the discussed fallacies and to satisfy the requirements formulated in Sect. 3, enabling new innovation competences in our Information Systems study programs, as well as an open-ended understanding of the benefits of Conceptual Modeling as a knowledge creation activity.

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.

4.2 Teaching Method and Course Flow

The teaching method is based on live tutorial demonstration of small implementation increments, mirrored by students. The progress has a "gradual revealing" nature, with metamodeling theorization provided in parallel lectures, to reflect back on the hands-on experience and by comparison with known modeling tools or languages. Each modeling method building block is showcased by a minimal example enriched across two iterations.

The tool development method employed for hands-on exercising is a simplification of the Agile Modeling Method Engineering (AMME) methodology. This is an iterative metamodeling approach where each iteration (i) starts with the definition of domain knowledge and modeling requirements (which here take the form of diagram mock-ups and the purpose of retrieving some information from models) and (ii) ends with the deployment of a usable modeling tool. More details on the AMME phases are available in [8], but for teaching purposes it is reduced here to its *Design* and *Develop* phases, quickly leading to a usable result even in the absence of introductory metamodeling theorization.

The two teaching iterations are exemplified in this paper, with the initial iteration satisfying the constraint of "not more than 3 concepts" and the second one splitting the modeling language into two types of models with machine-readable links, editable attributes and interactive notation. A third iteration may be left for students' homework, allowing them further individual exploration. The theoretical exposition that parallels the

hands-on experience follows a learning flow suggested in Fig. 2, where the argumentation cascades along the following steps:

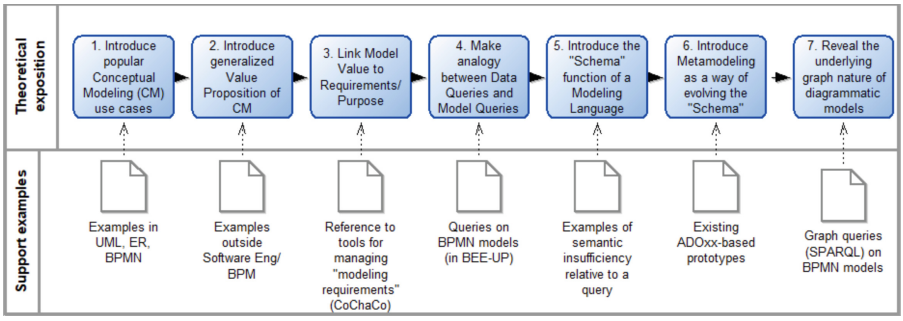

Fig. 2. Conceptual learning flow and support examples for each step

1. Introduce popular Conceptual Modeling use cases. This acts as a reminder of previous student experience from bachelor programs, where first contact with modeling involves the use of one or more of ER, UML, BPMN for the purpose of graphically documenting student projects;

2. Formulate a generalized "model value" proposition: first, by revealing (or reminding) that the models at the previous step may also have other purposes than graphical documentation - e.g. SQL code generation and BPMN process path simulation can be quickly demonstrated with open educational tools like BEE-UP [34]; next, the semantic coverage and domain-specificity of modeling languages is expanded towards other application areas and other purposes – e.g. Archimate, Value modeling, Designer components in Robotic Process Automation;

3. Link "model value" to Requirements/Purpose. This aims to make explicit the niche class of "modeling method requirements" (for which recent research proposed dedicated Requirements Engineering methods – see [35]). In the general sense, this reveals the nature of modeling languages/tools as Design Science artifacts subjected to requirements that trigger specific engineering cycles;

4. Make analogy between Data Queries and Model Queries to benefit from existing student experience with relational databases. Use the simplest examples that support the analogy – e.g. "select all tasks of a certain participant in a BPMN model". Use a tool that where model queries can be easily demonstrated directly in the modeling environment, e.g. BEE-UP [34];

5. Introduce the "schema" function of a modeling language. Reveal that model queries are enabled by a model schema ("metamodel"), similarly to how SQL queries are enabled by a database schema (and that the model schema must be sufficiently rich to satisfy the query). Provide examples of semantic insufficiency (e.g. how to retrieve ingredients and quantities from a BPMN diagram repurposed as a cooking recipe) – this will argue for the possibility of agile schema adaptation;

6. Introduce Metamodeling as a way of adapting the model schema to make it adequate for certain model queries (themselves derived from Requirements/Purpose). Show

existing prototypes of open modeling tools and present Metamodeling platforms as means for editing their "model schema" and for the rapid (re)prototyping of a modeling tool according to the changed schema;

7. Introduce the underlying graph-like nature of diagrammatic models and the possibility of enabling model queries outside a modeling environment. A handy example can again benefit from BEE-UP, as it provides an option to export BPMN or UML models to RDF graphs, making them available to semantic processing. This will also be shown in the teaching artifact developed with students, thus establishing a bridge towards the next course module (on semantic technology).

4.3 Initial Iteration of the Teaching Artifact

The teaching artifact is demonstrating starting with introducing the scenario (Sect. 4.1) immediately followed by the initiation of AMME's Design phase by (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 notions of metamodeling established in the MOF specification [20] to one easily involved in the data-models analogy.

Figure 3 shows such a 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; additionally, 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 generalized 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 in 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).

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 later 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

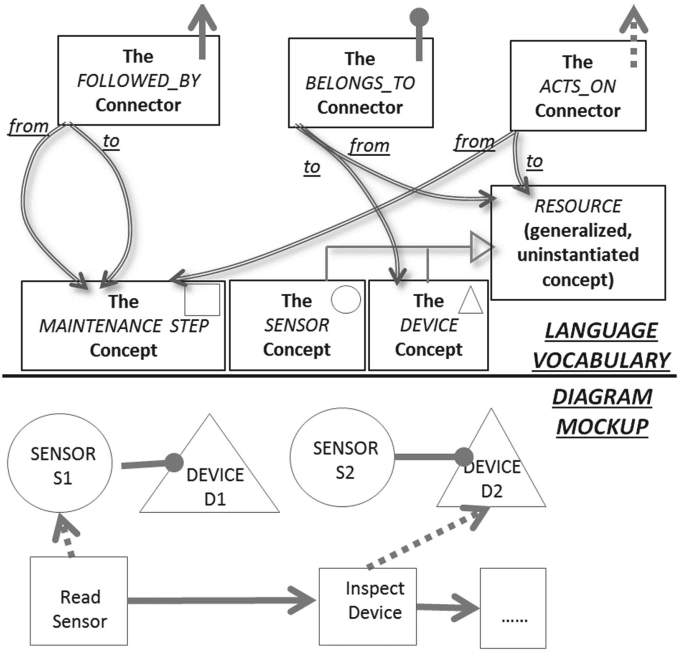

Fig. 3. Diagram mockup and derived language vocabulary

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.

Figure 4 shows a model created with the initial modeling tool implementation. The model is still limited in its capability of expressing information, relying on the labels attached to the concepts and not so much on the graphical representation. It is also very simplistic in terms of attributes that it exposes to model queries.

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 some 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 with the help of similarity

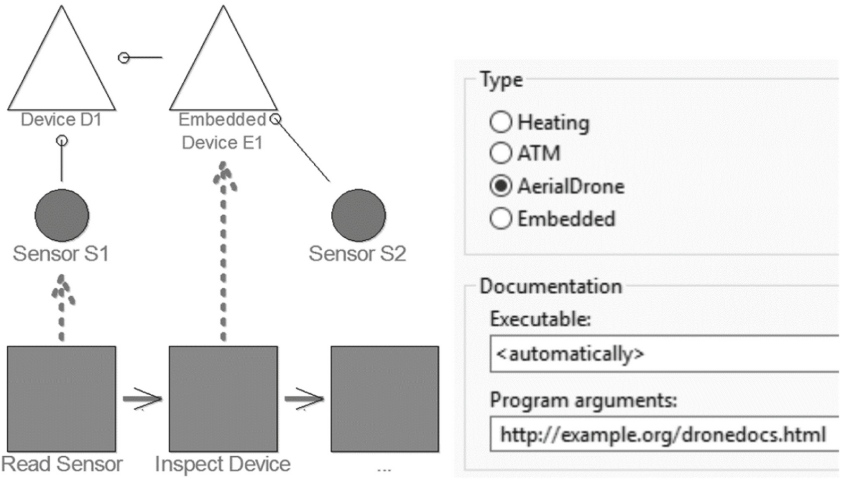

Fig. 4. Model created with the initial language iteration (left) and concept-level schema for DEVICE annotations (right)

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 [36] (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

4.4 Advanced Iteration of the Teaching Artifact

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