On Explaining Modeling Principles with Modeling Examples: A Classification Catalog

Martin Gogolla
University of Bremen, Germany
gogolla@informatik.uni-bremen.de

Antonio Vallecillo
University of Malaga, Spain
av@lcc.uma.es

ABSTRACT

Examples are of central concern in teaching modeling. Typically, the general principles and concepts that have to be communicated are explained in terms of smaller or larger modeling examples with the hope that the examples cover the central issues of the principles and concepts well. The paper discusses a classification catalogue for examples along various criteria like syntax, semantics, pragmatics, complexity or evolution. The aim of the paper is to encourage the teaching with examples exhausting the possible spectrum of example use.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General; K.3.0 [**Computers and Education**]: General

General Terms

Design, Languages, Documentation

Keywords

Teaching Modeling, Teaching with Examples, Example Quality, Example Adequancy, Relationship between Concept and Example

1. INTRODUCTION

Teaching a computer science subject means for us that general concepts and principles have to be explained to an interested audience. Such general and abstract ideas are usually made concrete by employing more or less convenient examples. The concepts live in the world of eternal thinking whereas the examples usually are alive in the finite world of doing. One might take the viewpoint that the concepts represent immortal gods wheras the examples have their manifestation as mere mortals.

When teaching modeling it is natural to introduce modeling concepts used at design time (e.g., class diagrams or state machines) and to manifest these concepts at runtime (e.g., by objects diagrams or runs through the state machine). Here, one can observe another reification of the God-Mortal duality: The general, abstract principles are determined at design time whereas the manifestations at runtime ideally obey the design time rules, or in other words, the mortals have to obey the laws determined by the gods.

Naturally, the teacher must make the choice for the treated concepts and principles and for the explaining examples and their presentation order: (a) concepts and principles may be mentioned first with explaining examples to follow, (b) the examples may come first and the general concepts and principles follow, or (c) concepts and principles as well as examples may be presented in interleaving order. However it is of central concern that examples must convey the general concepts and principles.

Our work has strong connections to other related papers. In [3] we have explained teaching model transformations with a larger, evolving UML/OCL example. Analogously to our approach, [7] and [11] point to the importance of using wrong resp. negative examples in teaching. [5] emphasizes the use of graphical modeling languages in teaching. [1] proposes an example for teaching refactoring techniques. [9] explains the popular Alloy method and tool with examples. The relationship between concepts and principles on the one hand and examples on the other hand is discussed in [10]. In [8] component technology is introduced by means of an example. Database modeling with SQL examples is the topic of [4] and [2].

The rest of the paper is structured as follows. First, we will introduce in Sect. 2 our criteria catalog. Second, we will explain the catalog with some selected examples in Sect. 3. The paper will be finished with concluding remarks in Sect. 4.

2. CONCEPTS AND PRINCIPLES FOR THE CLASSIFICATION CATALOGUE

This section introduces the different dimensions for example classification that we are proposing. The classification catalogue can be used during development of a teaching unit as a checklist to discover new examples or example variations and to assert that the character of the example collection is adequate to the aimed teaching purpose.

EduSymp'12, October 01-05 2012, Innsbruck, Austria

Complexity: Depending on the position of the respective modeling concept that is to be discussed, the example complexity may vary from *less complex* examples to *highly complex* ones. Typically one will show *introductory* examples first and move to *deepening* examples later.

Coverage degree of concepts and principles: All concepts and principles should be covered by examples in enough detail. Thus it makes sense to start with *overview* examples giving a general idea and to proceed to *detailed* examples that take up elements from the overview examples and explain all particularities in a sufficient and neccessary level.

Relevant software development phase: Modeling primarily concerns the requirement and the design phase, although modeling can be employed in all other development phases. Naturally, examples must be stated for the *design time*, however these examples must be taken up later and must be made concrete for the *runtime*. Also there may be separate characteristics between design time complexity and runtime complexity. A simple design time example may be illustrated with a highly complex runtime one or the other way round.

Syntax, semantics, pragmatics trinity: The syntax, semantics and pragmatics trinity [6] is of central concern for any language, in particular modeling languages. Examples should cover all different areas, i.e., *syntax*, *semantics* and *pragmatics*. There should be examples formulated with *canonical* syntactical constructs and examples formulated with *uncommon* syntactical constructs. The chosen examples should explain implications of *syntactical variations* for determination of the semantics. Last but not least *hypothetical* syntactical constructs that might be missing in the currently used modeling language but which are present in other languages could be discussed.

Criteria concerning example validity: Typically, one will show *positive*, i.e., syntactically well-formed, examples for the introduced concepts and might later also discuss *negative*, i.e., syntactically ill-formed, examples in order to distinguish between both categories [7, 11]. Other dualities relevant with respect to example validity are the dualities *wrong-correct* and *bad-good*. The wrong-correct duality refers to uses of language features that lead to semantically adequate or inadequate modeling situations. The bad-good duality points to questions concerning style and experience.

Evolution: An example may be an *a priori fixed* example or it may be a *dynamically evolving* or growing example, with or without audience involvement [3]. An example may be a recurring or *running* example whereas in particular situations *single-use* examples may be appropriate. Freshly introduced examples may alter with *variations* of previously used examples. The example can be *fully explained* at first occurrence or it may be *growing* during concept and principle elaboration.

Audience appropriateness: Examples may be tuned to be *audience specific* or they may be *generally useful* independent of a particular context.

Abstractness: Depending on the intended aim, examples can be understood to be *abstract* or they can refer to *concrete* situations. A similar, but different spectrum is span by *hypothetic* cases or, on the opposite side, by *real-world* case studies.

Assumptions: There are *safe* examples that work under all circumstances without any assumption. On the other hand, potentially *unsafe* examples assume particular implicit and usually unexpressed assumptions that have to be valid.

Example source: The examples may be taken from the *literature*, they can stem from own previous work or may be developed *freshly* for the teaching project under consideration.

3. EXAMPLES FOR INSTANTIATING THE CLASSIFICATION CATALOGUE

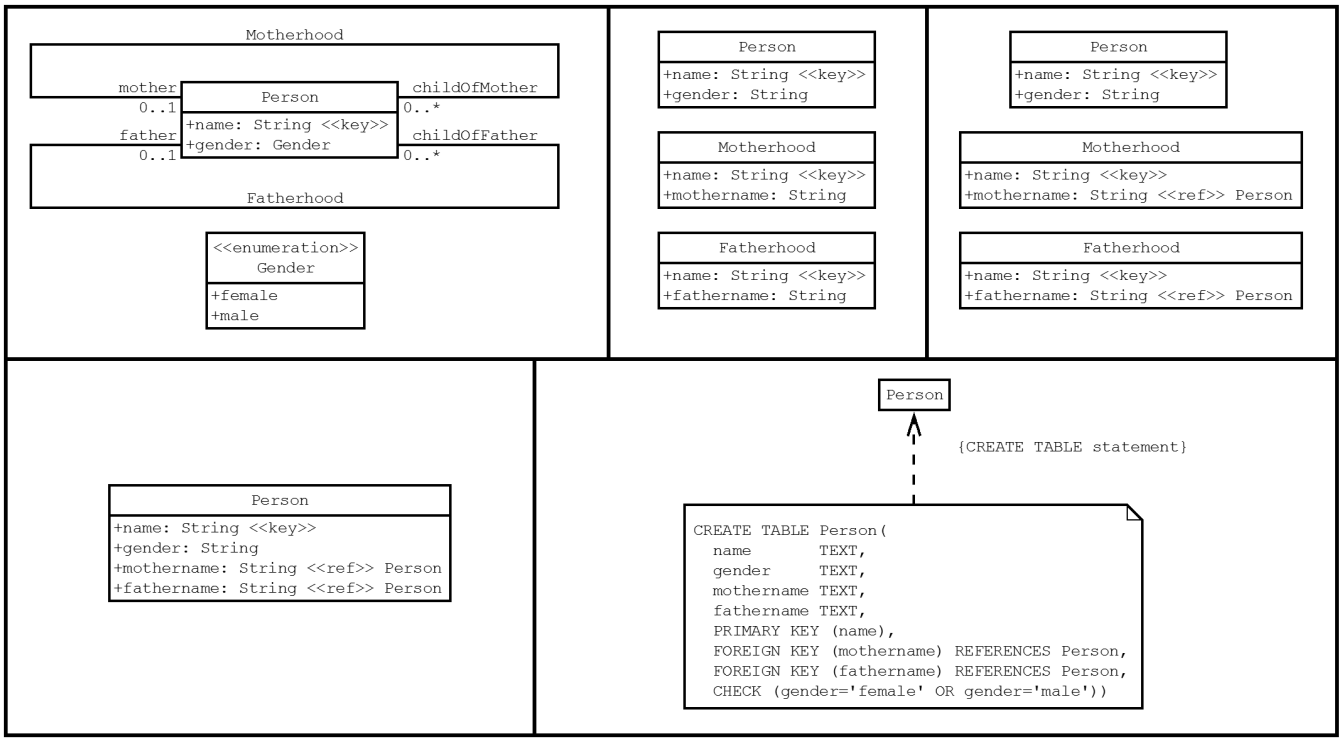
In Fig. 1 we picture the examples that we employ for illustrating our ideas. The upper part shows five UML class diagrams describing persons and parenthood relationships. The lower part displays three definitions of the same view: one view is formulated as a predicate in OCL, and two views are stated as SQL views. We now sketch our example classification catalogue w.r.t. this example, we will however not go into all interesting technical details of the examples.

Complexity: The starting UML class diagram is rather simple insofar that there is only one class, but there are two reflexive associations with four distinct rolenames. The final relational SQL schema concentrates the central SQL DDL concepts, i.e., tables, attributes, datatypes, primary keys, foreign keyes, and check constraints into a single table. We conclude that the example schemas are mildly complex due to the concentration of concepts.

Coverage degree of concepts and principles: If we assume that the example has to cover the basic UML concepts class, binary association and rolename and their transformation into a relational database schema, the basic concepts are covered well.

Relevant software development phase: The example is a classical example for the design phase. The relative mild complexity of the class diagram could be brought to a more involved complexity for the runtime, i.e., the database states. Having two alternative runtime models, i.e., one database state with some thousand persons all participating in the parenthood links and another database state with the same number of persons, but only very few participating in parenthood links, say only about ten links, one could nicely explain the difference between the class diagram 3 and class diagram 5: For the second database state there will very many null values for the representation of missing parent links.

Syntax, semantics, pragmatics trinity: A syntactic variation for the example class diagram would be to discuss what happens to the semantics if a requirement like 'Everybody has a mother, thus we must change the




```
GP_GC(gp:Person,gc:Person):Boolean=
  gp.childOfMother->union(gp.childOfFather)->
    collect(p|p.childOfMother->union(p.childOfFather))->
      includes(gc)
```

```
CREATE VIEW GP_GC AS                          -- GrandParent_GrandChild
SELECT gp.name AS gpname, gc.name AS gcname
FROM   Person gp, Person gc
WHERE  EXISTS (SELECT *                         -- maternal grandmother
               FROM   Person p
               WHERE  gp.name=p.mothername AND p.name=gc.mothername) OR
       EXISTS (SELECT *                         -- maternal grandfather
               FROM   Person p
               WHERE  gp.name=p.fathername AND p.name=gc.mothername) OR
       EXISTS (SELECT *                         -- paternal grandmother
               FROM   Person p
               WHERE  gp.name=p.mothername AND p.name=gc.fathername) OR
       EXISTS (SELECT *                         -- paternal grandfather
               FROM   Person p
               WHERE  gp.name=p.fathername AND p.name=gc.fathername)
```

```
CREATE VIEW GP_GC AS                          -- GrandParent_GrandChild
SELECT gp.name AS gpname, gc.name AS gcname
FROM   Person gp, Person gc
WHERE  EXISTS (SELECT *                         -- maternal grandmother
               FROM   Person p
               WHERE  gp.name=p.mothername AND p.name=gc.mothername) OR
       gp.name IN (SELECT p.fathername          -- maternal grandfather
                   FROM   Person p
                   WHERE  p.name=gc.mothername) OR
       gp.name =ANY (SELECT p.mothername        -- paternal grandmother
                     FROM   Person p
                     WHERE  p.name=gc.fathername) OR
       NOT ( gp.name <>ALL (SELECT p.fathername -- paternal grandfather
                            FROM   Person p
                            WHERE  p.name=gc.fathername) )
```

Figure 1: Different UML, OCL and SQL models in the Context of Motherhood and Fatherhood.

multiplicity from `0..1` to `1..1`' comes up during teaching. Furthermore, the second SQL view definition introduces all syntactically allowed SQL subquery forms in a condensed way. The semantics of the first SQL view definition is semantically equivalent to the second one. A discussion about syntax and semantics of SQL could be triggered in this example by discussing the syntactical well-formedness and semantical implications of changing the `SELECT *` subexpression in the second SQL view definition to `SELECT FALSE`. Generally, syntactic variations and their influence on semantics are fruitful discussion items.

Criteria concerning example validity: The shown second SQL view definition has a bad style insofar that it realizes the same functionality, i.e., checking for grandparent and grandchildren pairs, in four different ways.

Evolution: The example explains the evolution of a starting UML class diagram into a relational SQL schema. It starts with a conceptual model, steps through some intermediate schemas and ends in an implementation model.

Audience appropriateness: The example seems to be not very audience specific, and it should be easy to communicate it to a large audience. Depending on the audience, typical examples from the domain of the audience are welcomed.

Abstractness: The example in Fig. 1 is rather concrete. To give a case for a more abstract situation, the teaching unit under consideration might introduce the UML generalization constraint pairs *overlapping-disjoint* and *complete-incomplete*. In order to explain the possibiliy of combing these options in an orthogonal way, an abstract example (see Fig. 2) with a general superclass `G` and five subclasses `A, B, C, D, E` together with the four constraint options for the subclass pairs `{A, B}`, `{B, C}`, `{C, D}` and `{D, E}` could be given. Often, allowed syntactical options are better explained with abstract examples.

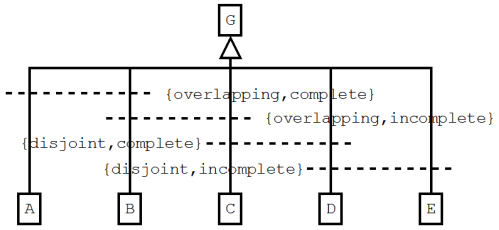


Figure 2: Abstract Example.

Assumptions: An implicit assumption that is not stated explicitly for the example is the assumption that only bodily parenthood links should be represented. This assumption is manifested with the multiplicity `0..1` on the parent side of the associations.

Example source: In this case the example is a combination of a classical association found in many textbooks and the need for explaining key SQL features, i.e., primary and foreign keys, in a very condensed way.

4. CONCLUSIONS

We think that teaching modeling and its central concepts and principles must be accompanied by various examples. We have proposed a criteria catalog for classifying the use of examples in teaching modeling. The catalog can be employed as a checklist during the development of a teaching unit in order to look for ways of integrating examples into the teaching unit. Our criteria catalog must be validated further and will be refined in upcoming teaching projects. Although examples are important, it is however essential to return from the example level to the concept and principle level, after having discussed concepts and principles as well as examples, and to explicitly point to the relationship between both. Thereby the connection between concepts and examples, between gods and mortals, is strengthened.

5. REFERENCES

[1] S. Demeyer, F. V. Rysselberghe, T. Gîrba, J. Ratzinger, R. Marinescu, T. Mens, B. D. Bois, D. Janssens, S. Ducasse, M. Lanza, M. Rieger, H. Gall, and M. El-Ramly. The lan-simulation: A refactoring teaching example. In *IWPSE*, pages 123–134. IEEE Computer Society, 2005.

[2] A. Fekete. Teaching transaction management with sql examples. In J. C. Cunha, W. M. Fleischman, V. K. Proulx, and J. Lourenço, editors, *ITiCSE*, pages 163–167. ACM, 2005.

[3] M. Gogolla. Teaching Touchy Transformations. In M. Smialek, editor, *MODELS Educators' Symposium (EDUSYMP'2008)*, pages 13–25. Warsaw University, ISBN 83-916444-8-0, 2008.

[4] S. Hartmann, M. Kirchberg, and S. Link. Design by example for sql table definitions with functional dependencies. *VLDB J.*, 21(1):121–144, 2012.

[5] M. Main and E. B. Koffman. Graphical examples for teaching fundamental cs1 concepts. In H. R. Arabnia, V. A. Clincy, A. Bahrami, and A. M. G. Solo, editors, *FECS*, pages 36–42. CSREA Press, 2010.

[6] A. Martinich, editor. *The Philosophy of Language*. Oxford University Press, 3rd edition, 1996.

[7] J. Pang, K. A. Yun, and M. Stoner. Use wrong examples as a tool for teaching. In H. R. Arabnia, V. A. Clincy, and N. Tadayon, editors, *FECS*, pages 217–221. CSREA Press, 2008.

[8] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, editors. *The Common Component Modeling Example: Comparing Software Component Models [result from the Dagstuhl research seminar for CoCoME, August 1-3, 2007]*. Springer, LNCS 5153, 2008.

[9] S. Tarkan and V. Sazawal. Chief chefs of z to alloy: Using a kitchen example to teach alloy with z. In J. Gibbons and J. N. Oliveira, editors, *TFM*, pages 72–91. Springer, LNCS 5846, 2009.

[10] T. van Gog. Effects of identical example-problem and problem-example pairs on learning. *Computers & Education*, 57(2):1775–1779, 2011.

[11] Z. Zhiying. Teaching software methodology with assistance of negative examples. In A. Abdelwahab and G. Rommel, editors, *ISCA Conference on Intelligent Systems*, pages 154–157. ISCA, 2001.