



7th Educators' Symposium @ MODELS 2011:
Software Modeling in Education
(EduSymp2011)

Model Correctness Patterns as an Educational Instrument

Azzam Maraee , Mira Balaban, Arnon Strum , Adiel Ashrov

10 pages

Model Correctness Patterns as an Educational Instrument

Azzam Maraee¹, Mira Balaban¹, Arnon Strum², Adiel Ashrov¹

¹ mari, mira, ashrov@cs.bgu.ac.il

Computer Science Department

Ben-Gurion University of the Negev, Beer-Sheva 84105, ISRAEL

² sturm@bgu.ac.il

Department of Information Systems Engineering

Ben-Gurion University of the Negev, Beer-Sheva, 84105, ISRAEL

Abstract: UML class diagrams play a central role in modeling activities. Given the difficulty in producing high quality models, modelers must be equipped with an awareness of model design problems and the ability to identify and correct such models. In this paper we observe the role of class diagram correctness patterns as an educational instrument for improving class diagram modeling. We describe a catalog of correctness and quality design (anti)-patterns for class diagrams. The patterns characterize problems, analyze their causes and provide repairing advice. Pattern specification requires an enhancement of the class diagram meta-model. The pattern classification has a major role in clarifying design problems. Finally, we describe an actual experiment of using the catalog for teaching modeling.

Keywords: Design patterns, correctness and quality patterns, educational instruments, model and Meta-Model levels, abstraction, visual language, pattern catalog.

1 Introduction

Models are the backbone of the emerging Model Driven Engineering (MDE) approach, whose major theme is development of software via repeated model transformations. The quality of models used in such a process affects not only the final result, but also the development process itself. In order to achieve high quality, it is important to have educated modelers, who are sensitive to model quality. Indeed, similarly to software construction, model quality can be improved by applying automatic transformations (refactoring), but this approach still cannot replace the need for good modelers.

Design patterns encapsulate expert advice for solving typical problems that might occur in multiple contexts. They fulfill an educational role: Awareness of design patterns yields better solutions. The design patterns trend in software construction gained increasing popularity since the appearance of the design patterns book of the “GoF” [GHJV95]. On the model-level, there is research on formulation of software patterns [LP09, BBC08b], formulation of model-level general and domain specific design anti-patterns [EBL11], proposals for design pattern specification languages [FKGS04, Kim07, BBC08a], and research on the impact of design patterns [TB11]. [EBL10] is a rich catalog of model-level patterns that identify modeling problems.

In this paper we present a catalog of modeling anti-patterns for problems of correctness and quality in class diagram design [BGU10], and discuss its role as an educational instrument, for

improving class diagram modeling. Given the widespread usage of UML class diagrams and the difficulty in producing high quality models, modelers must have deep understanding of model design problems, their identification and repair. Patterns of correctness and quality problems in modeling characterize typical situations in which correctness or quality problems arise, analyze the causes, and suggest possible solutions. Their educational role is to increase the awareness of designers to inter-relationships between modeling elements that create incorrect or low quality models.

To the best of our knowledge, this is the first catalog that provides an in-depth analysis of causes of correctness and quality problems, together with repair advices. The catalog is intended to play an educational role in teaching object modeling. In view of this goal, we discuss the pattern specification language, the problem-oriented organization of the catalog, and its instruction. Finally, we describe an actual experiment in using the catalog in teaching modeling.

Section 2 presents a variety of causes for incorrect class diagram modeling. Section 3 discusses the nature of the pattern specification language and presents an enhancement to UML class diagrams. Section 4 shortly introduces our patterns catalog, and Section 5 describes an experiment that observes the role of the correctness patterns in teaching class diagram modeling. Section 6 concludes the paper.

2 Correctness patterns

The two main correctness problems in UML class diagrams are *consistency* [BCG05] and *finite satisfiability* [MMB08]. Consistency deals with necessarily empty classes, and finite satisfiability deals with necessarily empty or infinite classes. Both problems are caused by problematic interaction of constraints.

Figure 1 presents four incorrect class diagrams, in which the kind of incorrectness problem, or the kind of problematic constraint interaction vary. Figure 1d presents an inconsistency problem, caused by the interaction of the disjoint generalization-set constraint, and the multiple inheritance of class C3: The disjoint constraint forces class C3 to be empty in every legal instance. Figures 1a, 1b, 1c present three cases of the finite satisfiability problem, caused by different kinds of constraint interaction. In Figure 1a each instance of C has a single successor and at least two predecessors. Therefore, if the number of C-s in a legal instance is c , and the number of predecessor-successor links is d , then d must satisfy $d = c \cdot 1$ and also $d \geq c \cdot 2$, implying the inequality $c \geq c \cdot 2$, that can be satisfied only by an empty or an infinite extension of class C. The problematic constraint interaction in this case involves the multiplicity constraints on the *predecessor* – *successor* association. In Figure 1b the problematic constraint interaction involves the multiplicity constraints in the cycle of associations w, q, r , and in Figure 1c the problem is with the class hierarchy between C1 and C, and the multiplicity constraints on the *parent* – *child* association.

Figure 1 shows that correctness problems are varied and can occur for many reasons. We argue that in real class diagrams it is not easy to understand the various interactions among constraints, and their impact on correctness or quality. Awareness to patterns that single out problematic constraint interactions, analyze the problems they create, and suggest possible repairs, improves the overall design quality.

[BGU10] present a catalog of correctness patterns, that sort out different templates of interac-

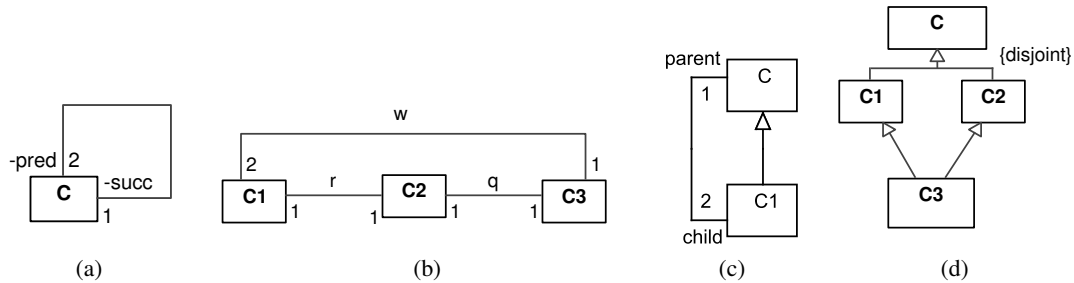


Figure 1: Finite satisfiability problems

tions, characterize the involved problems, and suggest solutions. Each pattern describes a design problem that is identified by characteristic structures within a class diagram [BMS10].

The class diagrams in Figures 1a, 1b are instances of the *Pure Multiplicity Cycle (PMC)* pattern, which characterizes finite satisfiability problems due to interaction of multiplicity constraints on a cycle of associations. Figure 2a informally sketches the identification template of this pattern. The dashed line indicates a sequence of successive binary associations (the binary associations path). The pattern includes an additional verification constraint, and analysis of possible repairs, like relaxing the multiplicity constraint “2” to “1..2”.

The class diagram in Figure 1c is an instance of the *Multiplicity Hierarchy Cycle (MHC)* pattern, which characterizes finite satisfiability problems due to interaction of multiplicity constraints on a cycle of associations and class hierarchy constraints. Figure 2b informally sketches the identification template of this pattern. The two dashed lines indicate an interleaved path of associations and class hierarchy constraints. This pattern also includes an additional verification constraint, and analysis of possible repairs, like switching the direction of a class hierarchy constraint in the cycle.

The class diagram in Figure 1d is an instance of the *diamond* pattern, which characterizes consistency problems due to interaction between multiple class hierarchy (multiple inheritance) and disjoint constraints. Figure 2c informally sketches the identification template of this pattern. A possible repair is to remove the disjoint constraint.

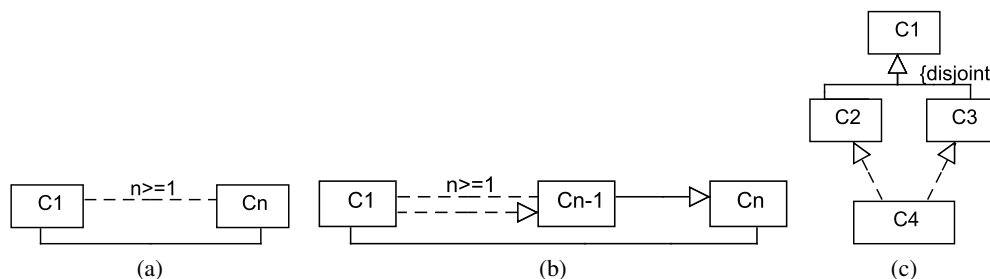


Figure 2: Informal sketches of identification templates of correctness patterns

Correctness patterns as an educational instrument

Using positive examples (like design patterns) or negative examples (like anti-patterns) is an effective educational tool for developing high modeling skills. [BSV09] show that positive exam-

ples enhance syntactic quality, while negative examples enhance semantics equality, and neither has much effect on pragmatic quality. While there is an educational value in presenting concrete examples, patterns provide an abstraction level that classifies the concrete cases along typical characterizations. [TB11] show the influence of design patterns on improving the inception of models (although their patterns are presented by concrete syntactic examples).

Correctness pattern abstraction characterizes correctness problems in terms of conceptual structures such as *cycle of associations* and *class hierarchy cycle*, rather than in terms of concrete examples. Furthermore, patterns provide an accurate specification of *problem domains*, and solution advices. They can be viewed as *anti-patterns* that point to negative designs and suggest repairs. The benefit is twofold: First, they provide a systematic approach for identifying pattern instances; second, they sharpen the distinction between different kinds of problematic constraint interaction.

3 Model-Pattern Specification Language

Pattern specifications include desired and undesired structures of elements from the pattern context. Writing a specification demands a language. Pattern writing approaches can be categorized by two factors: 1) textual notation vs. visual notation (or both); 2) expression at the model-level vs. expression at the Meta-Model-level. The main mode of usage for patterns is as expert advice for educational purposes. Formally specified patterns are also used as automatic transformations (refactorings). It seems that the appropriate mode for pattern expression depends on the intended usage.

Visual vs. textual specification: Visual notations are uniquely human-oriented representations, that facilitate human communication, comprehension and problem solving [Moo09]. In contrast, textual representation might provide a better support for rigorous reasoning but is inferior with respect to user comprehension [Kim07]. Indeed, pattern specifications usually employ some kind of visual representations, usually in an informal manner, using concrete examples. Good visual representations enjoy the *cognitive effectiveness* property, i.e., the ability to directly clarify translations between cognitive and visual concepts [FDCB10]. This property sets a criterion for visual language evaluation [Moo09].

Model-level vs. Meta-model-level specification: In the model-level approach for pattern specification, patterns are specified using typical examples, that are enhanced with textual specification [GHJV95, LP09]. This is the more popular approach, especially in software design patterns. Model-level specification proved helpful for communicating design experience to developers. Yet, since structures are expressed via examples and text, it can create ambiguities that make it difficult to verify conformance to patterns [KE07]. For example, Figure 1a does not capture the intentioned problem domain, i.e., “finite satisfiability problem due to a cycle of associations with multiplicity constraints” in a way that enables identifying it with the class diagram in Figure 1b (which belongs to the same problem domain). This approach does not lend itself to automation of reasoning about patterns.

Meta-Model based specification enables rigorous textual specification of problem domains that abstract concrete examples [EBL11, EBL10]. The two examples in Figures 1a and 1b can be captured by a single textual specification, demonstrated in Listing 1 in Section 4. Therefore,

meta-model-level specification can support pattern automation as refactorings. However, it is inappropriate for educational purposes, since defining and comprehending the meta-model-level textual patterns demand expert knowledge.

Conclusion: We argue that for educational purposes, patterns should have **visually**, hence **model-level** specification, but use a notation that captures the **meta-model-level abstractions**. For that purpose, the model-level visual language should be enhanced with new notation that enables visualization of the meta-level abstractions. Below, we describe few extended notations to the UML class diagrams, that enable visual, model-level specification of the correctness patterns in our catalog. A similar approach is used in [BBC08a].

Class diagram enhancement for pattern specification: The necessary abstraction involves constructs for specification of unbounded relationship structures like association paths, hierarchy paths, interleaved association and hierarchy paths, aggregation paths, etc. We extend the UML class diagram meta-model with new classes that capture these abstractions, and provide their concrete syntax as new visual notations in class diagrams.

Figure 3b presents the Meta-Model extension for the *generalization path* abstraction. The enhancement includes a new meta-class *GeneralizationPath*, and a derived meta-association *nextassoc*. Existing meta-model elements appear within dashed rectangles. The new visual notation for relationship paths uses $*$ for denoting paths of lengths ≥ 0 , and $+$ for paths whose lengths > 0 . These symbols are added as labels, on top of the standard relationship symbol. For example, a class hierarchy (generalization) path is represented by a generalization line, labeled by $*$. Figure 3a presents the identifying structure for the *Multiplicity-Hierarchy-Cycle* pattern, that uses the new generalization-path construct. The concrete syntax enhancement is demonstrated in Table 1. Figure 4 intuitively sketches a possible instantiation of the identification structure of the *Pure-Multiplicity-cycle* pattern.

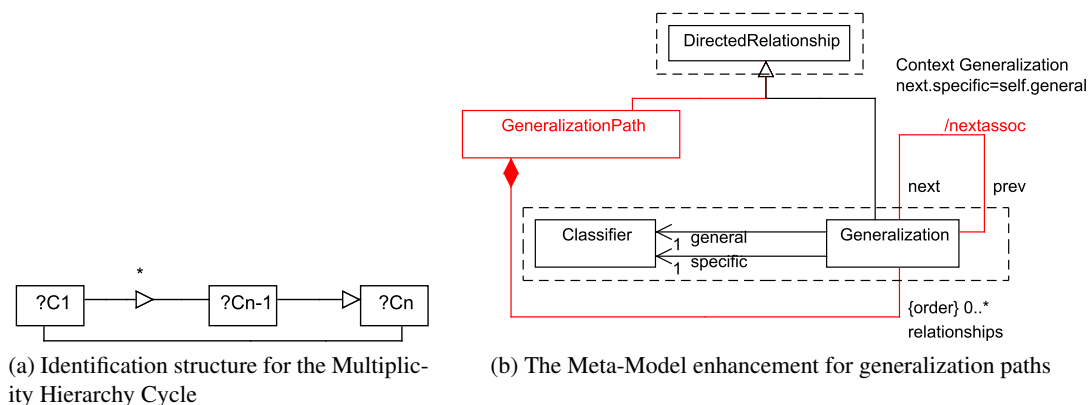


Figure 3

[Moo09] formulates nine evidence-based principles for achieving cognitively effective visual notations. Three main principles are: 1) *Semiotic Clarity principle* – importance of one-to-one correspondence between semantic constructs and graphical symbols; 2) *Semantic Transparency principle* – are symbols and their corresponding concepts are easily associated; 3) *Perceptual discriminability* – symbols that represent different constructs should be clearly distinguishable. We try to follow such principles in adopting new notation. In particular, the use of $*$, $+$ labeled


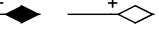
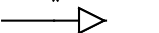

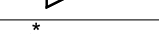
Graphic notation	Meaning
	A path of associations with length is ≥ 0
	A path of compositions (aggregation) with length ≥ 1
	A path of generalizations with length ≥ 0
	An interleaved path of associations and generalizations with length ≥ 0
	An interleaved path of compositions and generalizations with length ≥ 0

Table 1: Concrete syntax for relationship paths

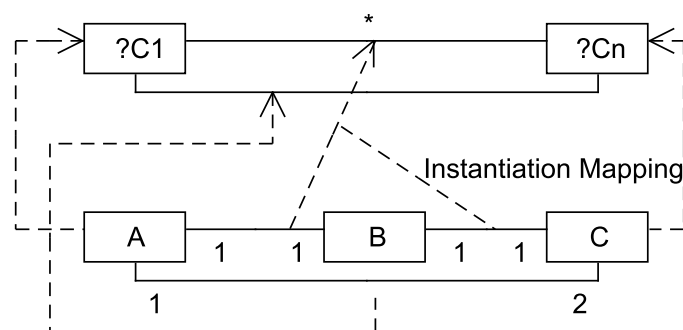


Figure 4: Instantiation of the Pure-Multiplicity-Cycle pattern

relationship lines is associated with the traditional meaning of these operators as denoting ≥ 0 and > 0 repetitions, respectively.

4 The Correctness-Pattern Catalog

The catalog includes patterns for solving problems of *correctness* or of *quality* of class diagrams. The correctness problems refer to the two formal correctness problems of consistency and finite satisfiability. Quality problems refer to formally correct design problems that do not meet criteria of desirable design. The quality problems are further classified into *incomplete design*, *redundancy problems*, and *comprehension problems*. Within the categories, patterns are classified by the kind of the constraint interaction that causes the problem. This problem based classification is contrasted with the approach of [EBL10], which is syntax-semantics-pragmatics based.

Based on the above classification the catalog currently includes a total of 45 patterns: 15 patterns for finite satisfiability problems, 11 patterns for *consistency problems* and 17 patterns for *quality* problems. The catalog is still under development, and new patterns are being added.

Pattern structure: Pattern specification is a template consisting of nine entries: 1. Name; 2. Pattern problem – A textual description of the problem handled by the pattern; 3. Concrete example; 4. Pattern identification structure – A class diagram snippet in the enhanced class diagram language; 5. Involved meta-model elements; 6. Pattern verification – A formal constraint imposed on the pattern identification structure that verifies occurrence of a problem; 7. Repair

advice (refactoring); 8. Related patterns; 9. Pattern justification – a correctness proof for the pattern identification, verification, and advice.

Example 1 (Pure-Multiplicity-Cycle Pattern)

1. **Pattern name:** *Pure Multiplicity Cycle (PMC)*.
2. **Problem:** A cycle of associations with multiplicity constraints might introduce a finite satisfiability problem.
3. **Concrete Example:** See Figure 5a.
4. **Pattern Identification Structure:** See Figure 5b.

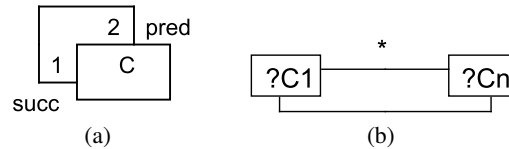


Figure 5

5. **Involved meta-model elements:** The meta-classes *Association* and *Class*.
6. **Pattern verification:** The pattern identification structure characterizes a necessary but not sufficient condition for existence of a finite satisfiability problem caused by the multiplicity constraints in an association cycle. The verification condition below provides the sufficient condition. Its specification requires an elaboration of the identification structure, as in Figure 6.

The cycle causes a finite satisfiability problem if one of the following conditions holds:

- $\prod_{i=1}^n m'_i < \prod_{i=1}^n n_i$.
- $\prod_{i=1}^n m_i < \prod_{i=1}^n n'_i$.

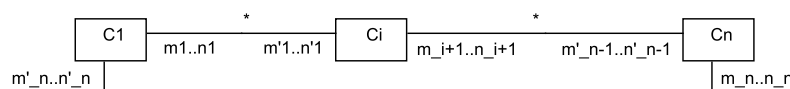


Figure 6: Pattern verification

7. **Repair advice:** Consider relaxation of the multiplicity constraints: decrease a minimal multiplicity value or increase a maximal multiplicity value.
8. **Related patterns:** The *Multiplicity Hierarchy Cycle (MHC)* pattern.
9. **Pattern justification:** Proof is omitted.

In order to emphasize the educational value of visual specification on the model level, we demonstrate, in Listing 1, the textual specification of the **PMC** identification structure in QVT, following [EBL10]¹. The specification finds all associations that participate in association cycles.

¹ The specification is based on their *Classifier Has Generalization Cycle* anti-pattern.


```

top relation AssociationCycle {
  checkonly domain source _assoc: Association{ allconnectedassoications ->
    includes(_assoc) {} };
  enforce domain target _c: Category {name = 'Anti-Patterns', pattern = _p:
    Pattern {name = 'AssocCycle', rootBinding = _rb1: RoleBinding {role = '
    Association', element = _assoc}}};}

```

Listing 1: A QVT based rule for detecting associations that participate in association cycles

5 Putting the catalog into practice

In order to examine the extent to which patterns help in identifying erroneous models, we conducted a series of experiments that check various factors that affect the usage of the correctness patterns. Table 2 presents the experiment settings: The subjects' background (Software Engineering and Information Systems Engineering students) and numbers, the education type (by simple example or by formal and abstract demonstration of the patterns), the domain type (real or synthetic domains), and the size of the domain and the number of errors.

Exp	Subjects	Education Type	Domain Type	Additional Factors	Checking Mode
1	SE (49)	Examples	Real	Different domain	Class Assignment
2	ISE (49)	Examples	Synthetic	Different size	Class Assignment
3	SE (42)	SE (42)	Synthetic		Class Assignment
4	SE (61)	Patterns	Synthetic		Exam

Table 2: Experiment settings

In the first experiment we divided the subjects into two groups. In the first session (before introducing the patterns) each group got a different domain model (via a class diagram) of an academic and a genome domain. In the second session (after introducing the patterns by examples by the paper authors), each group had the other domain model. In each session the subjects were asked to identify places in which correctness problems occur. The results of the experiment show that the differences among the subjects' achievements before and after introducing the patterns are not statistically significant (see Table 3). Examining the differences across the two domains does not show any statistical significance as well. Our conjecture is that the students focused on the domain semantics rather on the model semantics.

These results have led us to perform another experiment that was based on synthetic domains. Here again, we divided the subjects into two groups, yet the two groups had domain models with different sizes (with 6 and 2 problems, respectively). We had the same setting as in the first experiment, in which the subjects had to identify existing problems before and after introducing the patterns. The results show that in the case of a large domain model with several correctness problems, there is a significant improvement in identifying the problems after introducing the patterns (see Table 3).

The third experiment followed the same setting, but the pattern introduction was more thorough, and we provided the subjects with a comprehensive explanation of the notion of correctness patterns, their formalism, abstraction and the pattern catalog. In this experiment we examined the improvement in one group and indeed, we found that the achievements made by the subjects

Exp	Before	After
1	42% (Academia); 49% (Genome);	39%(Genome); 48%(Academia)
2	28% (6 patterns) ; 52% (2 patterns)	51% (6 patterns); 50% (2 patterns)
3	42%	70%
4	Precision = 85.57%, Recall = 66.39%	

Table 3: Experiment results

were high (70 % identification of the existing problems) compared to previous experiments (see Table 3).

In the fourth examination, we checked the extent to which the patterns help in identifying only existing problems. For that purpose we calculated the recall measure which is the ability of the subjects to identify all relevant problems and the precision recall which is the ability of the subjects to identify only the relevant problems. The results (as appear in Table 3) show that the patterns indeed provide support for identifying only relevant problems. Yet, their support for identifying all relevant problems is limited.

Summarizing the results we found out that the correctness patterns indeed provide guidelines for identifying erroneous models and that their effectiveness increases in cases where the domain models are complex (and thus have more problems) and when their introduction is presented in a more formal and abstract form (instead of by examples).

Pattern instruction: Teaching catalogs of patterns or refactorings is hard [Pil10, Kop11]. This is a well known problem, that results from the monotonic repetitive character of catalogs. Not surprisingly, we have found through our experiments that we have been most successful when we taught only one or two patterns out of each category. The problem oriented organization of the catalog encourages a focused usage, such as when a new kind of constraint is introduced.

6 Conclusion

This paper presented a catalog of correctness and quality patterns for class diagram design, and discussed its role as an educational instrument. We argued that for educational purposes patterns should be visually formulated, using model-level concepts, and provided an appropriate enhancement to UML class diagram.

The correctness catalog is still under development. About 30 additional patterns are planned. In the future, we plan to include the catalog as a routine class material in our object modeling courses.

Bibliography

- [BBC08a] D. Ballisa, A. Baruzzo, M. Comini. A Minimalist Visual Notation for Design Patterns and Antipatterns. In *Fifth International Conference on Information Technology: New Generations*. Pp. 51–56. 2008.
- [BBC08b] D. Ballisa, A. Baruzzo, M. Cominia. A Rule-based Method to Match Software Patterns Against UML Models. *Electronic Notes in Theoretical Computer Science* 219:51–66, 2008.

- [BCG05] D. Berardi, D. Calvanese, D. Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence* 168:70–118, 2005.
- [BGU10] BGU Modeling Group. UML Class Diagram Patterns. 2010.
<http://www.cs.bgu.ac.il/~cd-patterns/>
- [BMS10] M. Balaban, A. Maraee, A. Sturm. Management of Correctness Problems in UML Class Diagrams – Towards a Pattern-Based Approach. *International Journal of Information System Modeling and Design* 1(1):24–47, 2010.
- [BSV09] N. Bolloju, C. Schneider, D. Vogel. Asymmetrical Effects of Using Positive and Negative Examples on Object Modeling. In *18th International Conference on Information Systems Development*. 2009.
- [EBL10] M. Elaasar, L. Briand, Y. Labiche. Metamodeling Anti-Patterns. 2010.
<https://sites.google.com/site/metamodelingantipatterns>
- [EBL11] M. Elaasar, L. Briand, Y. Labiche. Domain-Specific Model Verification with QVT. In *ECMFA 2011*. 2011.
- [FDCB10] K. Figl, M. Derntl, M. Caeiro Rodriguez, L. Botturi. Cognitive effectiveness of visual instructional design languages. *Journal of Visual Languages & Computing*, 2010.
- [FKGS04] R. France, D. Kim, S. Ghosh, E. Song. A UML-Based Pattern Specification Technique. *IEEE Transactions on Software Engineering* 30(3):193–206, 2004.
- [GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Volume 206. Addison-wesley Reading, MA, 1995.
- [KE07] D. Kim, C. El Khawand. An approach to precisely specifying the problem domain of design patterns. *Journal of Visual Languages & Computing* 18(6):560–591, 2007.
- [Kim07] D. Kim. The Role-Based Metamodeling Language for Specifying Design Patterns. In Taibi (ed.), *Design Pattern Formalization Techniques*. Pp. 183–205. IGI Global, 2007.
- [Kop11] C. Koppe. A Pattern Language for Teaching Design Patterns. In *European conference on pattern languages of programs, EuroPLoP 2011*. 2011.
- [LP09] M. Llano, R. Pooley. UML Specification and Correction of Object-Oriented Anti-patterns. In *ICSEA '09*. Pp. 39–44. 2009.
- [MMB08] A. Maraee, V. Makarenkov, B. Balaban. Efficient Recognition and Detection of Finite Satisfiability Problems in UML Class Diagrams: Handling Constrained Generalization Sets, Qualifiers and Association Class Constraints. In *MCCM08*. 2008.
- [Moo09] D. Moody. The ”physics” of notations: towards a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering* 35(6):756 – 779, 2009.
- [Pil10] N. Pillay. Teaching Design Patterns. In *Southern African Computer Lecturers Association Conference, SACLA 2011*. 2010.
- [TB11] O. Tanriover, S. Bilgen. A framework for reviewing domain specific conceptual models. *Computer Standards & Interfaces* 33(5):448 – 464, 2011.