# Measuring complexity of logical puzzles with metrics expressed in OCL *or* A case study on OCL diversity*

Martin Gogolla[1], Jesús Sánchez Cuadrado[2]

[1]University of Bremen
[2]Universidad de Murcia

**Abstract**

Logical puzzles can support teaching informatics and rational thinking. We have developed a metamodel-based approach supporting development, configuration, and automatic construction of puzzles with a syntax close to predicate logic and OCL and a semantics close to UML object models. (1) We extend this approach by introducing puzzle metrics expressed with OCL that consider puzzle complexity w.r.t. syntactic and semantic aspects. The aim of developing the metrics is to tune the logical puzzles for novice and expert players and learners. (2) Our UML and OCL model presents a new case study on the power and flexibility of OCL because OCL is here applied for various purposes and on different modeling levels.

**Keywords**

Logical puzzle, Rational thinking, Puzzle complexity, Puzzle metrics

## 1. Introduction

For teaching rational thinking and computer science, logical puzzles are considered as motivating instruments. They typically introduce a realistic problem on which one can elaborate abstract concepts. Puzzles are useful for improving problem solving skills, and they may improve engagement and motivation in the teaching process.

We have applied modeling techniques to the automated development of logical puzzles supporting teaching methods for computer science and mathematics, but also for other disciplines. We have developed an automatic approach [1] for formulating and solving logical puzzles. The focus was on teaching modeling and formal methods to university students, but we also took the perspective to apply puzzles to teach elementary formal reasoning to school pupils. For expressing models, we employ the Unified Modeling Language (UML [2]) and the Object Constraint Language (OCL [3, 4]). As the underlying modeling tool we apply the system USE (UML Specification Environment, [5, 6]). As a central component, USE offers a model finder that can automatically construct object models for a class model with OCL constraints. The original approach [1] was here refactored and simplified (clearer, shorter operations and invariants) in order to achieve more compact puzzle handling and is in this paper newly extended with metrics for puzzles; a new aspect here is that, OCL model invariants are divided clearly into aspects of puzzle syntax and aspects of puzzle semantics. Our paper presents also a new case study for the power of OCL on different modeling levels and tasks. Roughly, we develop a metamodel and a model in a "Requirements-Design-Implementation-Testing" process and use OCL on the metamodel and model level and in most process phases.

## 2. Puzzle metamodel

In Fig. 1 our class model for representing logical puzzles is shown. Logical puzzles use clues and properties about imagined things that are to be described. Among the possible things, there is often

*STAF 2025 Workshops: OCL, OOPSLE, LLM4SE, ICMM, AgileMDE, AI4DPS, and TTC. Koblenz, Germany, June 10-13, 2025*
*Corresponding author.
†These authors contributed equally.
gogolla@uni-bremen.de (M. Gogolla); jesusc@um.es (J. S. Cuadrado)

one puzzle solution that is uniquely determined by the stated clues. A clue can ban things from being the solution. In our approach, the class model possesses a syntax part, classes `Clue` and `Property`, and a semantic part, the class `Thing` and the association between `Clue` and `Thing` with which a `Clue` is connected to a set of `banned Thing` objects, and a `Thing` is connected to a set of `banning Clue` objects. A central ingredient of the class model is the operation `Thing::Valid(c:Clue)` that computes whether a `Clue` evaluates to `true` or to `false` in a `Thing`. The class model can be understood best by discussing an object model that instantiates the classes. Figure 2 shows a puzzle with cards representing clues and a solution in progress indicated with tokens.

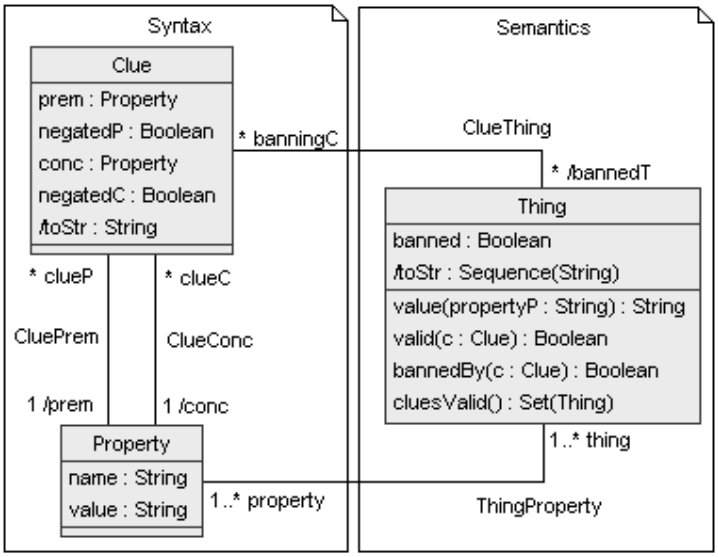


**Figure 1:** Class model for puzzle metamodel.

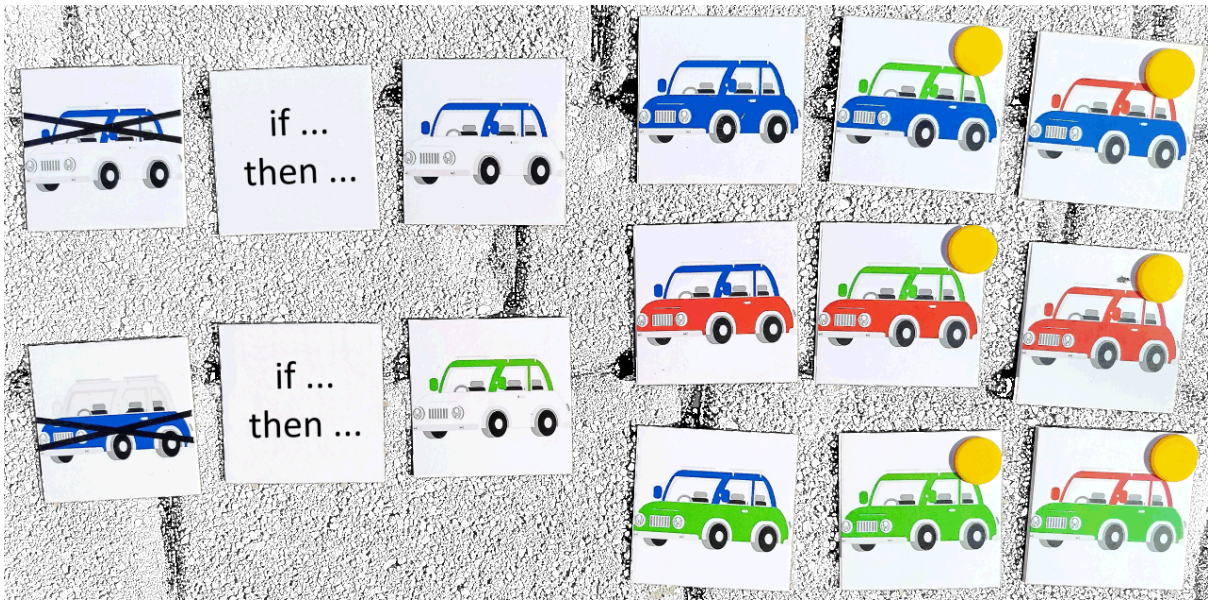


**Figure 2:** Possible puzzle representation and partial solution; yellow tokens mean banned by first clue.

In our approach, clues take the form of an implication with a premise and a conclusion, both being a comparison between a property and a value `Prop[=|<>]Val: PremProp[=|<>]PremVal => ConcProp[=|<>]ConcVal`. Fig. 3 shows an example puzzle in which, roughly speaking, the two `Property` types `top` and `bottom` can take the three values `red`, `green` and `blue` as represented by

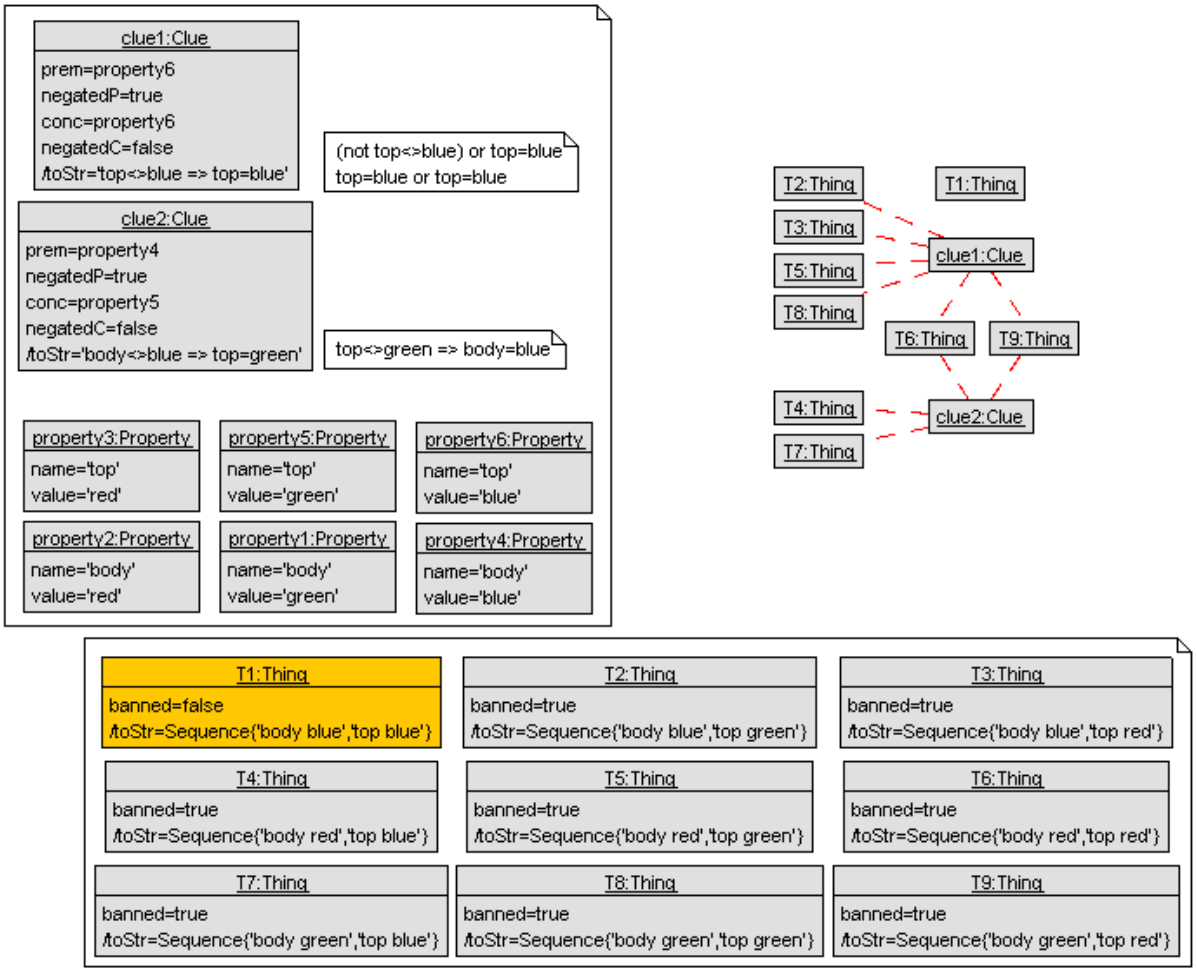


**Figure 3:** Object model for example puzzle.

the six `Property` objects. Two `Clue` objects are stated as implications: `top<>blue => top=blue` and `body<>blue => top=green`. This puzzle is automatically generated and will appear later in Fig. 4 in the first group of puzzles (with two clues) as `Puz5`. The nine `Thing` objects establish all puzzle solution candidates. The unique puzzle solution is the yellow marked `Thing` that is the only one where the attribute `banned` takes the value `false`. In the upper right, the (semantic) relationship between clues and things is pictured: A clue can ban a set of things, and things can be banned by a set of clues. The fact that `T1` is the puzzle solution is reflected by the circumstance that `T1` is not banned by any clue. Other links are not shown in the object diagram, however links can be deduced from the derived attributes `toStr` that are displayed in `Clue` and `Thing` objects.

All details of the model can be found in the Appendix. In particular all OCL expressions used as invariants, as operation definitions or as derivation rules for derived attributes or derived association ends can be found there. It is a rather compact, small model represented on basically only two pages. As central model features, we mention that (a) the operation `Thing::valid(c:Clue)` specifies whether a clue holds in a thing, and (b) the existence of a unique puzzle solution is formulated declaratively by the invariant `oneThingWithCluesValidNotBanned`.

## 3. Metrics for Puzzles

Our metrics are divided into OCL expressions referring to the syntax part (referring to class `Clue`) and to the semantics part of the class model (referring to class `Thing`). The expressions about syntax count the number of occurrences of particular patterns in the clues; the expressions about semantics count

clues, in which the things that are banned by the clue, possess particular properties. In more detail, we have developed the following seven metrics, i.e., OCL expressions. In the following we will partly refer to the metrics by their occurrence number in this list. E.g. metric 2 will refer to the expression about the number of ground facts.

1. Number of negations in clues
```
Clue.allInstances->select(c|c.negatedP=true)->size+
Clue.allInstances->select(c|c.negatedC=true)->size
```
2. Number of ground facts
```
Clue.allInstances->
  select(c|c.prem=c.conc and c.negatedP=true and c.negatedC=false)->size
```
3. Number of clues with different areas in premise and conclusion
```
Clue.allInstances->select(c|c.prem.name<>c.conc.name)->size
```
4. Number of clues banning nothing
```
Clue.allInstances->select(c|c.bannedT->isEmpty)->size
```
5. Number of clues banning exactly one thing
```
Clue.allInstances->select(c|c.bannedT->size=1)->size
```
6. Number of clues with equivalents
```
Clue.allInstances->select(c1|Clue.allInstances->exists(c2|
  c1<>c2 and c1.bannedT=c2.bannedT))->size
```
7. Number of indispensable clues
```
let C=Clue.allInstances in C->select(c1|c1.bannedT->exists(t|C->forAll(c2|
  c1<>c2 implies c2.bannedT->excludes(t))))->size
```

The first three metrics concentrate on the syntactic form of the clues and on its components, e.g. whether a clue contains a particular language feature. These metrics count: (1) the number of negations in the clues, (2) the number of clues representing de facto ground facts (i.e., whether the clue has the form `not(A) implies A` which is equivalent to `not(not(A)) or A` which again is equivalent to `A or A` and thus to `A`) and (3) the number of clues where premise and conclusion consider different property types. The last four metrics treat the semantics of clues and the connection between the clue and the set of banned `Thing` objects: (4) the number of clues without effect in the sense that the clue does not ban any possible `Thing` object, (5) the number of clues banning exactly one `Thing` object, (6) the number of clues possessing among all clues an equivalent different clue (banning the same set of `Thing` objects) and (7) the number of clues being indispensable in the sense that the clue bans a `Thing` object that is not banned by another clue. Typical metrics for software rely on the syntactic form of the underlying program text. In our approach we have explicitly modeled semantics through the class `Thing` and an appropriate association with class `Clue`, and thus we can take into account the effect, the semantics, that a clue has. We regard this as a unique feature of our approach that allows us to make well-argued statements about the complexity of a clue and with this whether a clue is well-suited for a novice or more suited for an expert.

We have validated that the metrics can be meaningfully applied with the running example introduced earlier. With the USE model validator we have generated 36 puzzles: 9 puzzles with 2 clues, 9 with 3 clues, 9 with 4 clues and 9 with 5 clues; the circumstance that we consider 9 puzzles comes from the fact that we have generated in each group all possible solutions for top and body color on the basis of 3 different colors ($3 * 3 = 9$). The table in Fig. 4 shows in each row a puzzle and in the columns the result of evaluating the metrics. Additionally, in the last column all metric values are summed up. As expected, with growing number of clues the puzzle complexity increases.

However, in each group with an equal number of clues, there are also differences in the puzzle complexity between the various considered metrics. Let us discuss some details of the table. For example, in the 2-group (the puzzles with 2 clues) the metric 6 (Equivalents) must always evaluate to 0: if there are only 2 clues and the puzzle offers only 2 properties (`top` and `body`), there cannot be equivalent clues; similar arguments could be stated for metrics 4 (Banning Nothing) and 5 (Banning

| | | Nega-tions | Ground Facts | Different Areas | Banning Nothing | Banning One | Equi-valents | Indispen-sables | SUM |
|---|---|---|---|---|---|---|---|---|---|
| 2 clues | Puz1 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| | Puz2 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| | Puz3 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| | Puz4 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| | Puz5 | 2 | 1 | 1 | 0 | 0 | 0 | 2 | 6 |
| | Puz6 | 2 | 2 | 0 | 0 | 0 | 0 | 2 | 6 |
| | Puz7 | 2 | 1 | 1 | 0 | 0 | 0 | 2 | 6 |
| | Puz8 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| | Puz9 | 3 | 1 | 1 | 0 | 0 | 0 | 2 | 7 |
| 3 clues | Puz1 | 5 | 0 | 2 | 0 | 0 | 0 | 3 | 10 |
| | Puz2 | 4 | 1 | 2 | 0 | 0 | 0 | 2 | 9 |
| | Puz3 | 4 | 0 | 1 | 0 | 0 | 0 | 3 | 8 |
| | Puz4 | 3 | 0 | 3 | 0 | 0 | 0 | 3 | 9 |
| | Puz5 | 5 | 0 | 2 | 0 | 0 | 0 | 3 | 10 |
| | Puz6 | 4 | 0 | 3 | 0 | 0 | 0 | 3 | 10 |
| | Puz7 | 3 | 1 | 2 | 0 | 0 | 0 | 3 | 9 |
| | Puz8 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 6 |
| | Puz9 | 5 | 1 | 2 | 0 | 0 | 0 | 2 | 10 |
| 4 clues | Puz1 | 6 | 0 | 2 | 0 | 0 | 0 | 3 | 11 |
| | Puz2 | 6 | 0 | 1 | 1 | 0 | 0 | 3 | 11 |
| | Puz3 | 3 | 0 | 3 | 0 | 1 | 0 | 3 | 10 |
| | Puz4 | 6 | 0 | 3 | 0 | 0 | 0 | 4 | 13 |
| | Puz5 | 6 | 1 | 1 | 1 | 0 | 0 | 2 | 11 |
| | Puz6 | 4 | 1 | 1 | 2 | 0 | 2 | 2 | 12 |
| | Puz7 | 6 | 0 | 3 | 0 | 0 | 0 | 3 | 12 |
| | Puz8 | 4 | 1 | 2 | 1 | 0 | 0 | 2 | 10 |
| | Puz9 | 6 | 0 | 2 | 1 | 0 | 0 | 3 | 12 |
| 5 clues | Puz1 | 5 | 0 | 1 | 0 | 1 | 2 | 3 | 12 |
| | Puz2 | 6 | 1 | 1 | 2 | 0 | 2 | 2 | 14 |
| | Puz3 | 9 | 0 | 4 | 0 | 0 | 0 | 4 | 17 |
| | Puz4 | 7 | 1 | 1 | 1 | 0 | 0 | 1 | 11 |
| | Puz5 | 6 | 0 | 3 | 1 | 0 | 0 | 3 | 13 |
| | Puz6 | 4 | 0 | 3 | 1 | 0 | 0 | 4 | 12 |
| | Puz7 | 6 | 0 | 4 | 0 | 0 | 0 | 3 | 13 |
| | Puz8 | 1 | 0 | 4 | 0 | 0 | 0 | 4 | 9 |
| | Puz9 | 9 | 1 | 3 | 0 | 0 | 0 | 2 | 15 |

**Figure 4:** Metrics for 36 generated puzzles.

One) since these metrics also evaluate to 0 in the 2-group. A further interesting point is that in the 5-group there is a wide range of use of negations (metric 1) ranging from clue sets with only 1 negation to clue sets with 9 negations. As the puzzles are automatically generated, there are also random decision taken by the USE model validator in the construction: E.g., in the 5-group, there are 2 puzzles with 2 equivalent clues, but there could also be 3 or 4 puzzles with that feature.

The metrics can even help to tune the syntactic representation of the clues, say, for novices or experts. If the metric on ground facts is non-zero, then there exists a clue in the form `not(A) => A`. For an expert, one could present the clue in this form; for a novice, one has the option to present the clue simply as `A`. In the running example, this would be, for example, a syntax representation like `top<>red => top=red` vs. a syntax representation like `top=red`.

The seven metrics that we have proposed are intended to demonstrate the options in our approach. A lot of further metrics on puzzle complexity could be formulated. For example, one could look for clue tuples in which the set of banned `Thing` objects of the first clue is strictly included in the set of banned `Thing` objects of the second clue. Or, one could look for clues that only ban `Thing` objects that are not banned by another clue.

## 4. Use of OCL for different purposes and levels

OCL expressions are applied in our approach for different purposes and on different modeling levels (Fig. 5).

1. OCL expressions are part of the class model as class invariants (for syntax, semantics, and efficiency), operation definitions and derivation rules for derived attributes and association ends.
2. Clues may be thought of as class invariants in an imagined UML model representing the puzzle. In the running example, one could think of the puzzle in terms of a class `Car` with attributes `top:String`, `body:String` and the invariants `clue1: top<>'blue' implies top='blue'` and `clue2: body<>'blue' implies top='green'`. Finding an object model for this specification with fitting attribute values then corresponds to finding a solution for the logical puzzle.
3. When applying the Model Validator, we use classifying terms for generating many diverse object models each representing a puzzle solution. The classifying terms that we use are:

   ```
   SolutionTopColor
     let TC=Thing.allInstances->any(t | Clue.allInstances->
           forAll(c | t.valid(c))).property->any(p|p.name='top').value in
     if TC='red' then 1 else if TC='green' then 2 else 3 endif endif
   SolutionBodyColor
     let BC=Thing.allInstances->any(t | Clue.allInstances->
          forAll(c | t.valid(c))).property->any(p|p.name='body').value in
     if BC='red' then 1 else if BC='green' then 2 else 3 endif endif
   ```

   These two classifying terms take care that each two generated object models differ in at least one of the classifying terms. For a fixed number of clues, one obtains nine object models with different `(top,body)` color values from the range `red, green, blue`.
4. A further application of OCL lies in the main focus of our proposal, namely the use of OCL for formulating metrics that measure the puzzle complexity with regard to syntactic and semantic aspects.

OCL is applied in the original model in a declarative way. Furthermore, OCL is used in the sequential construction process together with the Model Validator in a way that guarantees that particular object models are found. And, in connection with the metrics, OCL is applied to optimize the user involvement and interaction.

## 5. Related work

For a full discussion of related work see [1]. We here only give a short list of references.

Puzzles in education: The use of puzzles is a well-known approach to teaching of problem solving skills and mathematical thinking [7, 8]. For this kind of puzzles both solvers [9] and builder [10] tools have been proposed. Puzzles have been used to teach first-order logic. In [11] a variant of teaching first-order logic is used in which the system also generates new instances of the game automatically.

Solving puzzles: The topic of solving puzzles has traditionally been addressed by using solvers of different types (e.g., SAT or SMT solvers). For instance, in [12] several puzzles along with their encoding in a logic formalism are discussed. LLMs were be used as an interface to interact with the puzzle [13].

Developing puzzles: Some DSLs have been proposed to specify puzzles and their gaming context. For instance, EGGG [14] is DSL which allows the user to encode games, and it generates an interface to play with them. In [15] a DSL is proposed to encode human strategies for solving logical puzzles like Sudoku or Nonograms. Ortiz et al. [16] propose a system to generate Nonograms puzzles based on color images.

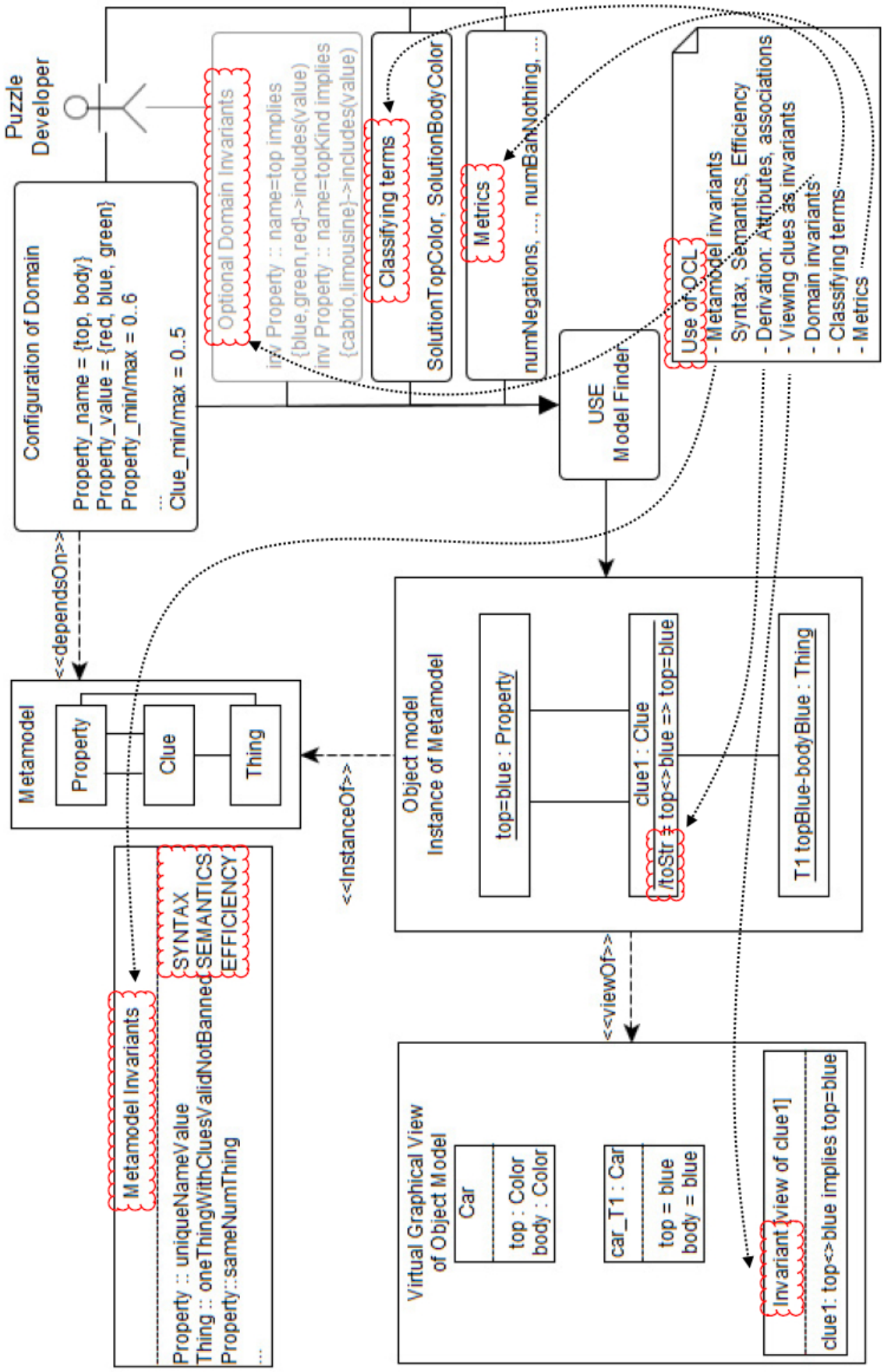


**Figure 5:** Applications of OCL in our approch.

The industry of logical puzzles: Although logical puzzles have a a tradition dating back many centuries [17] and many well known logical puzzles have been created (e.g., the *zebra puzzle* also known as Einstein's riddle), there is still an active industry which produces new puzzles and publishes them in magazines, books and dedicated websites. The website https://www.brainzilla.com/ offers a large number of puzzles of different kinds.

## 6. Conclusion and Future Work

This contribution has refactored and extended an existing approach for the construction of logical puzzles. We have added various metrics for puzzles that allow to tune the puzzles, for example, for software novices or experts. The contribution has also highlighted the various options for application and use of OCL in the approach.

Future work to be done includes extending the form of clues that can be specified and polishing the existing implementation. On the side of the metrics, other options than the ones proposed here should be studied. In the polished implementation, the metrics should be incorporated automatically so that appropriate, different clues are presented, for example, to novices or experts.

## Acknowledgement

Excellent reviewer remarks have improved the paper, although due to page limitations and our idea for paper structure we could not implement all. But we will respect the remarks in future work.

## References

[1] M. Gogolla, J. Sanchez Quadrado, Developing configurations and solutions for logical puzzles with UML and OCL, Software and Systems Modeling (2025). To appear. https://tinyurl.com/4djaks9e.

[2] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 2nd Edition, 2004.

[3] J. Warmer, A. Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, Addison-Wesley, 2nd Edition, 2004.

[4] J. Cabot, M. Gogolla, Object Constraint Language (OCL): A definitive guide, in: M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.), Proc. 12th Int. School Formal Methods for the Design of Computer, Communication and Software Systems: Model-Driven Engineering, Springer, Berlin, LNCS 7320, 2012, pp. 58–90.

[5] M. Gogolla, F. Büttner, M. Richters, USE: A UML-Based Specification Environment for Validating UML and OCL, Journal on Science of Computer Programming, Elsevier NL 69 (2007) 27–34.

[6] M. Gogolla, F. Hilken, K. Doan, Achieving model quality through model validation, verification and exploration, Comput. Lang. Syst. Struct. 54 (2018) 474–511. URL: https://doi.org/10.1016/j.cl.2017.10.001. doi:10.1016/J.CL.2017.10.001.

[7] E. F. Meyer, N. Falkner, R. Sooriamurthi, Z. Michalewicz, Guide to teaching puzzle-based learning, Springer, 2014.

[8] N. Falkner, R. Sooriamurthi, Z. Michalewicz, Teaching puzzle-based learning: development of basic concepts, Teaching Mathematics and Computer Science 10 (2012) 183–204.

[9] A. N. Kumar, Epplets: A tool for solving parsons puzzles, in: T. Barnes, D. D. Garcia, E. K. Hawthorne, M. A. Pérez-Quiñones (Eds.), 49th ACM Tech. Symp. CS Education, SIGCSE 2018, ACM, 2018, pp. 527–532.

[10] E. Deitrick, Graphical parsons puzzle creator: Sharing the full power of 2d parsons problems through a graphical, open-source online tool, in: M. Sherriff, L. D. Merkle, P. A. Cutter, A. E. Monge, J. Sheard (Eds.), SIGCSE'21: 52nd ACM Tech. Symp. CS Edu., 2021, ACM, 2021, p. 1379.

[11] A. Groza, M. M. Baltatescu, M. Pomarlan, Minefol: A game for learning first order logic, in: IEEE 16th Int. Conf. Intel. Comp. Communication and Processing (ICCP), IEEE, 2020, pp. 153–160.

[12] A. Groza, Modelling Puzzles in First Order Logic, Springer Nature, Cham, Switzerland, 2021.
[13] A. Groza, C. Nitu, Natural language understanding for logical games, arXiv preprint arXiv:2110.00558 (2021).
[14] J. Orwant, Eggg: Automated programming for game generation, IBM Systems Journal 39 (2000) 782–794.
[15] E. Butler, E. Torlak, Z. Popović, Synthesizing interpretable strategies for solving puzzle games, in: Proc. 12th Int. Conf. Foundations Digital Games, 2017, pp. 1–10.
[16] E. G. Ortíz-García, S. Salcedo-Sanz, J. M. Leiva-Murillo, Á. M. Pérez-Bellido, J. A. Portilla-Figueras, Automated generation and visualization of picture-logic puzzles, Computers & Graphics 31 (2007) 750–760.
[17] J. Rosenhouse, Games for Your Mind: The History and Future of Logic Puzzles, Princeton University Press, 2022.

## A. Class model including operation definitions, derivations and constraints (OCL elements only)

```
class Thing
attributes
  banned:Boolean
  toStr:Sequence(String) derive:
    self.property->iterate(p;
      r:Sequence(String)=Sequence{}|
      r->including(p.name+' '+p.value))->sortedBy(e|e)
operations
value(propertyP:String):String=
  self.property->any(p|p.name=propertyP).value

valid(c:Clue):Boolean =
  let pv:String=self.value(c.prem.name) in
  let pc:String=c.prem.value            in
  let cv:String=self.value(c.conc.name) in
  let cc:String=c.conc.value            in
  if c.negatedP=false and c.negatedC=false then (pv=pc  implies cv=cc ) else
  if c.negatedP=false and c.negatedC=true  then (pv=pc  implies cv<>cc) else
  if c.negatedP=true  and c.negatedC=false then (pv<>pc implies cv=cc ) else
  if c.negatedP=true  and c.negatedC=true  then (pv<>pc implies cv<>cc) else
     false endif endif endif endif

bannedBy(c:Clue):Boolean = not(self.valid(c))

cluesValid():Set(Thing) =
  Thing.allInstances->select(t |
    Clue.allInstances->forAll(c | t.valid(c)))

end

association CluePrem between
  Clue     [0..*] role clueP
 Property [1..1] role prem derive = Property.allInstances->any(p | self.prem=p)
end
```

```
association ClueConc between
  Clue     [0..*] role clueC
 Property [1..1] role conc derive = Property.allInstances->any(p | self.conc=p)
end

association ClueThing between
  Clue  [0..*] role banningC
  Thing [0..*] role bannedT derive =
    Thing.allInstances->select(t | t.bannedBy(self))
end

class Clue
attributes
  prem:Property
  negatedP:Boolean -- negated premise, e.g. *top<>blue* => bottom=red
  conc:Property
  negatedC:Boolean -- negated conclusion, e.g. bottom=red => *top<>blue*
  toStr:String derive:
    prem.name+if negatedP then '<>' else '=' endif+prem.value+' => '+
    conc.name+if negatedC then '<>' else '=' endif+conc.value
end

constraints

-- Constraints restricting syntactic elements
context p1,p2:Property inv uniqueNameValue:
  p1<>p2 implies (p1.name<>p2.name or p1.value<>p2.value)
context t1,t2:Thing inv uniqueProperty:
  t1<>t2 implies t1.property<>t2.property
context t:Thing inv uniquePropertyName:
  t.property->forAll(p1,p2 | p1<>p2 implies p1.name<>p2.name)
context c1,c2:Clue inv uniqueClue: c1<>c2 implies
  ( c1.prem<>c2.prem         or c1.conc<>c2.conc         or
    c1.negatedP<>c2.negatedP or c1.negatedC<>c2.negatedC )

-- Constraints restricting semantic elements
context Thing inv thingsWithCluesValid_EQ_thingsNotBanned:
  cluesValid()=Thing.allInstances->select(t | t.banned=false)
context Thing inv oneThingWithCluesValidNotBanned:
  cluesValid()->select(t | t.banned=false)->size=1

-- Constraints for Model Validator efficiency
context t1,t2:Thing inv sameNumProperty:
  t1.property->size=t2.property->size
context p1,p2:Property inv sameNumThing:
  p1.thing->size=p2.thing->size
```

## Declaration on Generative AI

The author(s) have not employed any Generative AI tools.