Experimenting with Multi-Level Models in a Two-Level Modeling Tool

Martin Gogolla

Database Systems Group, University of Bremen, Germany
gogolla@informatik.uni-bremen.de

Abstract. This paper discusses two ways to establish the connection between two levels in a multi-level model. The first approach uses normal associations and generalizations under the assumption that the multi levels are represented in one UML and OCL model. The second approach views a middle level model both as a type model and as an instance model and introduces operations to navigate between the type and instance level. The paper reports on some experiments that have been carried out.

Keywords. UML, OCL, Model, Multi-level model, Metamodel, Level connection.

1 Introduction

Metamodeling has become a major topic in software engineering research [6, 7, 16]. There are, however, a lot of discussions about central notions in connection with metamodels like *potency* or *clabject* where no final conceptual definition has been achieved. On the other hand, software tools for metamodeling are beginning to be developed [9, 2].

Metamodeling is closely connected to multi-levels models because the instantiation of a metamodel is a model. That model, when viewed as a type model, may be instantiated again and can then be viewed as an instance model. Proceeding this way, at least three model levels arise.

This paper discusses two ways to establish the connection between two levels in multi-level models. The first approach uses ordinary associations and generalizations under the assumption that the multi levels are represented in one UML and OCL model. The second approach views a middle level model both as a type model and as an instance model and introduces operations in order to navigate between the type and instance level. The paper reports on some experiments that have been carried out. The first approach joins the metamodels of several levels into one model as in our previous work [12]. Details about the first approach can be found in [13]. The second approach has not been put forward in a paper yet.

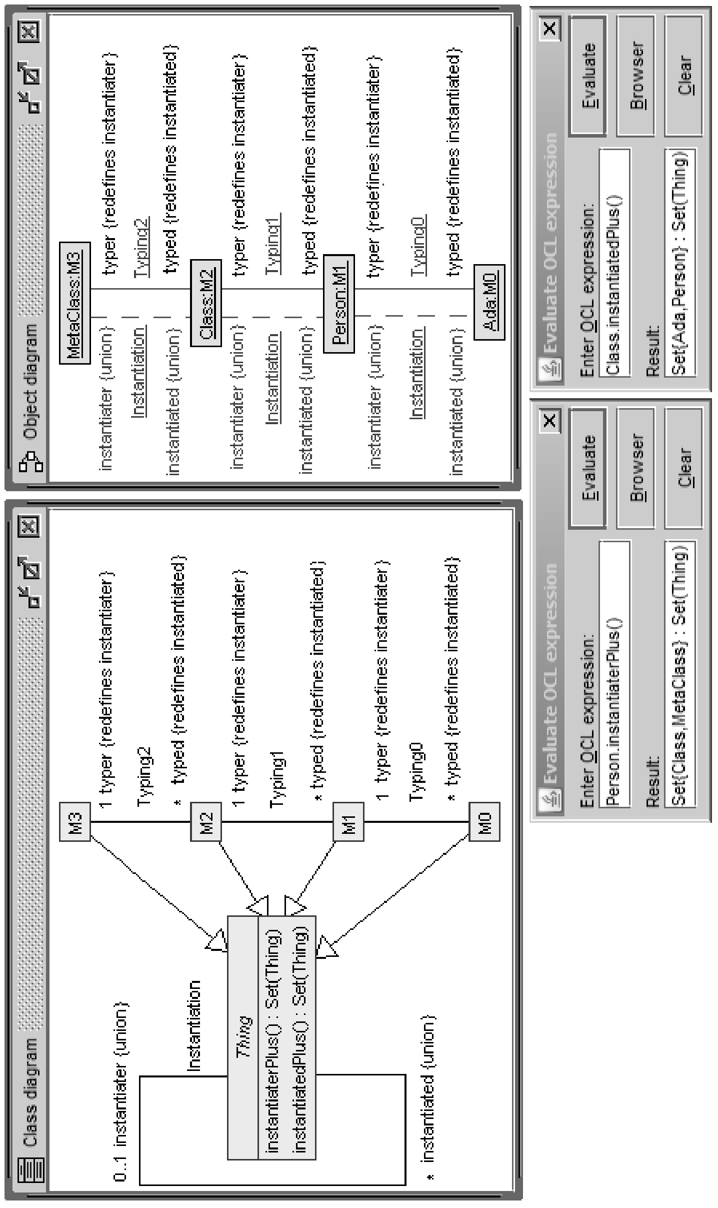
Our work has links to other related or similar approaches. The tool Melanie [2] is designed as an Eclipse plug-in supporting strict multi-level metamodeling and support for general purpose as well as domain specific languages. Another tool is MetaDepth [9] allowing linguistic as well as ontological instantiation with an arbitrary number of metalevels supporting the potency concept. In [16] the authors describe an approach to flatten metalevel hierarchies and seek for a level-agnostic metamodeling style in contrast to the OMG four-layer architecture. The approach in [14] also transforms multi-level models into a two-level model and focusses on constraints. Similar to our approach employing UML and OCL, the work in [17] uses F-Logic as an implementation basis for multi-level models including constraints. A conceptual clean foundation for multi-level modeling is discussed in [8]. [4] studies model constraints and [3] discusses multi-level connections in presence of a multi-level architecture distinguishing between a linguistic and an ontologic view on modeling.

The structure of the rest of the paper is as follows. Section 2 gives a small example for establishing the multi-level connection with associations and generalizations. Section 3 discusses the second approach that uses operations to connect the multi-levels. The contribution is closed with a conclusion and future work in sect. 4.

2 Connecting Multi-Levels with Associations and Generalizations

The first approach connects multi-levels with usual model elements: associations and generalizations. The example in Fig. 1 shows a substantially reduced and abstracted version of the OMG four-level metamodel architecture with modeling levels M0, M1, M2, and M3. Roughly speaking, the figure states: `Ada` is a `Person`, `Person` is a `Class`, and `Class` is a `MetaClass`. The figure does so by formally building an object diagram for a precisely defined class diagram including an OCL invariant that requires cyclefreeness when constructing instance-of connections. The distinction between `MetaClass` and `Class` is that when `MetaClass` is instantiated something is created that can be instantiated on two lower levels whereas for `Class` instantiation can only be done on one lower level. The model has been formally checked with the tool USE [11, 10]. In particular, we have employed the features supporting UML generalization constraints as discussed in [1, 15].

Concepts on a respective level M_x are represented in a simplified way as a class M_x. All classes M_x are specializations of the abstract class `Thing` whose objects cover all objects in the classes M_x. On that abstract class `Thing` one association `Instantiation` is defined that is intended to represent the instance-of connections between a higher level object and a lower level: an object of a lower level is intended to be an instance of an object on a higher level. The association `Instantiation` on `Thing` (with role names `instantiater` and `instantiated`) is employed for the definition of the associations `Typing0`, `Typing1`, and `Typing2` between M_x and M_{x+1} all having roles `typer` and `typed`. The role `typer` is a redefinition of `instantiater`, and `typed` is a redefinition of `instantiated`. The multiplicity `1` of `typer` narrows the multiplicity `0..1` of `instantiater`.


Class diagram
Thing
instantiaterPlus() : Set(Thing)
instantiatedPlus() : Set(Thing)
0..1 instantiater {union}
Instantiation
* instantiated {union}
M3
1 typer {redefines instantiater}
Typing2
* typed {redefines instantiated}
M2
1 typer {redefines instantiater}
Typing1
* typed {redefines instantiated}
M1
1 typer {redefines instantiater}
Typing0
* typed {redefines instantiated}
M0
Object diagram
MetaClass:M3
instantiater {union}
typer {redefines instantiater}
Instantiation
Typing2
instantiated {union}
typed {redefines instantiated}
Class:M2
instantiater {union}
typer {redefines instantiater}
Instantiation
Typing1
instantiated {union}
typed {redefines instantiated}
Person:M1
instantiater {union}
typer {redefines instantiater}
Instantiation
Typing0
instantiated {union}
typed {redefines instantiated}
Ada:M0
Evaluate OCL expression
Evaluate
Browser
Clear

```
Enter OCL expression:
Person.instantiaterPlus()
Result:
Set{Class,MetaClass} : Set(Thing)
```

```
Enter OCL expression:
Class.instantiatedPlus()
Result:
Set{Ada,Person} : Set(Thing)
```

Fig. 1. Ada, Person, Class, MetaClass within Single Object Diagram.

In the abstract class Thing the transitive closure instantiatedPlus() of instantiated is defined by means of OCL. Analogously, instantiaterPlus() is defined for instantiater. The closure operations are needed to define an invariant in class Thing requiring Instantiation links to be acyclic.

```
abstract class Thing
operations
  instantiatedPlus():Set(Thing)=
    self.instantiated->closure(t|t.instantiated)
  instantiaterPlus():Set(Thing)= ...
constraints
  inv acyclicInstantiation: self.instantiatedPlus()->excludes(self)
end
```

The class diagram from the left side of Fig. 1 is made concrete with an object diagram on the right side. The fact that the three associations Typing0, Typing1, and Typing2 are all redefinitions of association Instantiation is reflected in the object diagram by the three dashed links for association Instantiation with common role names instantiater and instantiated (dashed links in contrast to continuous links for ordinary links). Viewing Instantiation as a generalization (in terms of redefinition) of all $\texttt{Typing}_x$ associations allows to use the closure operations from class Thing on objects from classes M0, M1, M2 or M3. Thus the displayed OCL expressions and their results reflect the following facts: object Person is a (direct resp. indirect) instantiation of objects Class and MetaClass; objects Ada and Person are (direct resp. indirect) instantiations of object Class.

Metamodeling means to construct models for several levels. The metamodels on the respective level should be described and modeled independently (e.g., as M0, M1, M2, and M3). The connection between the models should be established in a formal way by a typing association (e.g., Typing0 gives a type object from M1 to a typed object from M0). The Typing associations are introduced as redefined versions of the association Instantiation from (what we call) a multi-level *superstructure*. This superstructure contains the abstract class Thing which is an abstraction of all metamodel elements across all levels and additionally contains the association Instantiation and accompanying constraints. Because Instantiation is defined as union, an Instantiation link can only connect elements of adjacent levels, i.e., the $\texttt{Typing}_x$ links are level-conformant and strict. The aim of the devices in the superstructure is to establish the connection between metamodel levels in a formal way and to provide support for formally restricting the connections.

3 Connecting Multi-Levels with Operations

The second approach for connecting multi-levels is based on viewing one model both as a type model and as an instance model. In Fig. 2 an example as elaborated with USE [11, 10] is shown. The object diagram on the left side is essentially equivalent to the class diagram on the right side. The displayed example can be realized in USE in this way, however, there is currently no option to connect the two equivalent representations in form of the class diagram and the object diagram. Additional features would be needed.

In Fig. 4 such a connection is indicated with (a) one operation accessing a model element through its String-valued name `$_$ : String -> ModelElement` and (b) one operation returning the String-valued name of a model element `#_# : ModelElement -> String`.

Apart from the two new operations dollar `$` and sharp `#` some new modeling features for UML and OCL would be needed as well: (a) OCL clauses (for elements such as invariants or pre- and postconditions) should allow parameters that represent variables for model elements and (b) the new operations dollar and sharp should be allowed in expressions for model elements in clauses (e.g., for a class or an attribute). The following examples try to give an impression of the aimed functionality.

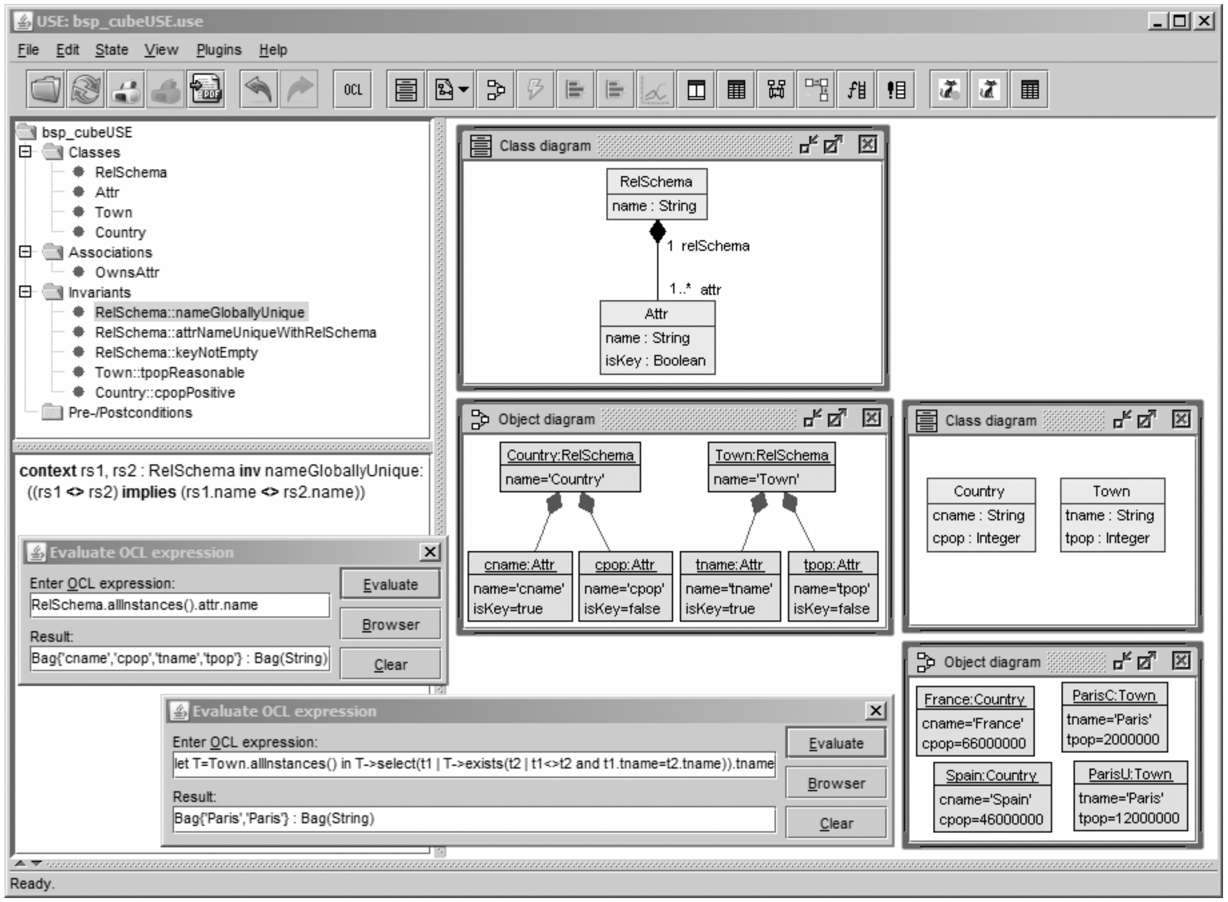
```
parameter[rs:RelSchema]
let relSchemaClass = $rs.name$ in -- in this example: $rs.name$=rs
let keyAttr = $rs.attr->any(a|a.isKey=true).name$ in
context relSchemaClass inv keyAttrUnique:
  relSchemaClass.allInstances->forAll(x,y |
    x<>y implies x.keyAttr<>y.keyAttr)

parameter[rs:RelSchema]
let relSchemaClass = $rs.name$ in
let keyAttr = $rs.attr->any(a|a.isKey=true).name$ in
context x,y:relSchemaClass inv 'keyAttrUniqueIn' + #rs#:
  x<>y implies x.keyAttr<>y.keyAttr
```

In these examples it is assumed that there is exactly one key attribute for each class. When these parameterized features become actualized, then in this case the following invariants would be required. It is an open question whether the actualization is implicit or explicit: The actualization mechanism may be considered as being implicit for all actual features that are available in the model, or the actualization mechanism may be considered as being something the modeler has to explicitly ask for.

```
context Town inv keyAttrUnique:
  Town.allInstances->forAll(x,y | x<>y implies x.name<>y.name)
context Country inv keyAttrUnique:
  Country.allInstances->forAll(x,y | x<>y implies x.name<>y.name)

context x,y:Town inv keyAttrUniqueInTown:
  x<>y implies x.name<>y.name
context x,y:Country inv keyAttrUniqueInCountry:
  x<>y implies x.name<>y.name
```



context rs1, rs2 : RelSchema **inv** nameGloballyUnique:
((rs1 **<>** rs2) **implies** (rs1.name **<>** rs2.name))

Enter OCL expression:
RelSchema.allInstances().attr.name
Result:
Bag{'cname','cpop','tname','tpop'} : Bag(String)

Enter OCL expression:
let T=Town.allInstances() in T->select(t1 | T->exists(t2 | t1<>t2 and t1.tname=t2.tname)).tname
Result:
Bag{'Paris','Paris'} : Bag(String)

Fig. 2. Example for Multiple Representations of Middle Level Model.

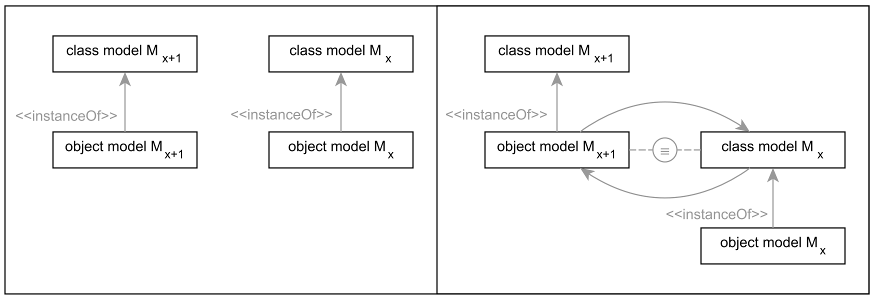


Fig. 3. General Scheme for Multiple Representations of Middle Level Model.

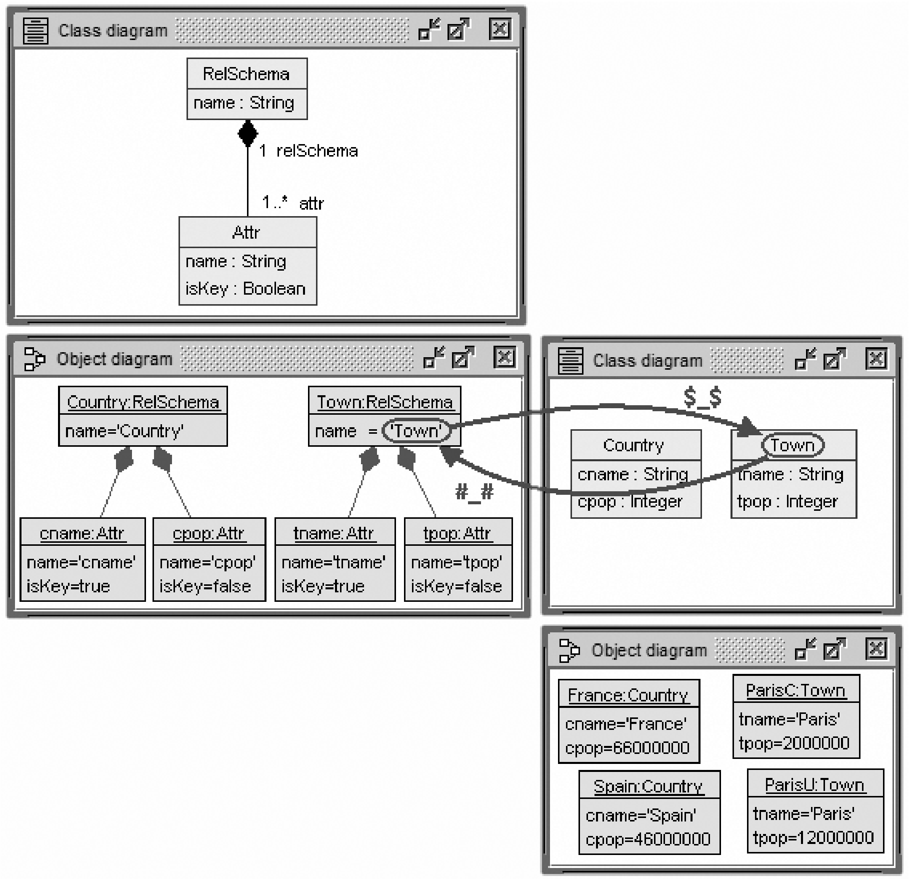


Fig. 4. Operations between Multi-Levels.

4 Conclusion

This paper is based on the idea to describe different metamodels in one model and to connect the metamodels either (a) with generalizations and associations employing appropriate UML and OCL constraints or (b) with special operations allowing to navigate between different multi-levels in a model. The paper does not claim to present final results, but some experiments and ideas in the context of multi-level models.

Future research includes the following topics. We would like to work out for our approach formal definitions for notions like potency or strictness. The notion of powertype will be given special attention in order to explore how far this concept can be integrated. Our tool USE could be extended to deal with different metamodel levels simultaneously. So far USE deals with class and object diagram. In essence, we think of at least a three-level USE (cubeUSE) where the middle level can be seen at the same time as an object and class diagram, as has been sketched in the second approach in this paper. Furthermore, larger examples and case studies must check the practicability of the proposal.

References

1. Alanen, M., Porres, I.: A Metamodeling Language Supporting Subset and Union Properties. Software and System Modeling **7**(1) (2008) 103–124
2. Atkinson, C.: Multi-Level Modeling with Melanie. Commit Workshop 2012 (2012) commit.wim.uni-mannheim.de/uploads/media/commitWorkshop_Atkinson.pdf.
3. Atkinson, C., Gerbig, R., Kühne, T.: A Unifying Approach to Connections for Multi-Level Modeling Foundations. In: Proc. Int. Conf. Models (MODELS'2015). (2015)
4. Atkinson, C., Gerbig, R., Kühne, T.: Opportunities and Challenges for Deep Constraint. In: Proc. Int. Workshop OCL and Textual Modeling (OCL'2015). (2015)
5. Atkinson, C., Grossmann, G., Kühne, T., de Lara, J., eds.: Proc. MODELS'2014 Workshop on Multi-Level Modelling (MULTI'2014). Volume 1286 of CEUR Workshop Proceedings., CEUR-WS.org (2014)
6. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. IEEE Software **20**(5) (2003) 36–41
7. Bézivin, J.: On the Unification Power of Models. Software and System Modeling **4**(2) (2005) 171–188
8. Clark, T., Gonzalez-Perez, C., Henderson-Sellers, B.: A Foundation for Multi-Level Modelling. [5] 43–52
9. de Lara, J., Guerra, E.: Deep Meta-modelling with MetaDepth. In Vitek, J., ed.: TOOLS (48). Volume 6141 of LNCS., Springer (2010) 1–20
10. Gogolla, M.: Employing the Object Constraint Language in Model-Based Engineering. In Gorp, P.V., Ritter, T., Rose, L., eds.: Proc. 9th European Conf. Modelling Foundations and Applications (ECMFA 2013), Springer, LNCS 7949 (2013) 1–2
11. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming **69** (2007) 27–34

12. Gogolla, M., Favre, J.M., Büttner, F.: On Squeezing M0, M1, M2, and M3 into a Single Object Diagram. In et al., T.B., ed.: Proc. MoDELS'2005 Workshop Tool Support for OCL and Related Formalisms, EPFL (Switzerland), LGL-REPORT-2005-001 (2005)
13. Gogolla, M., Sedlmeier, M., Hamann, L., Hilken, F.: On Metamodel Superstructures Employing UML Generalization Features. In Atkinson, C., Grossmann, G., Kühne, T., de Lara, J., eds.: Proc. Int. Workshop on Multi-Level Modelling (MULTI'2014), http://ceur-ws.org/Vol-1286/, CEUR Proceedings, Vol. 1286 (2014) 13–22
14. Guerra, E., de Lara, J.: Towards Automating the Analysis of Integrity Constraints in Multi-Level Models. [5] 63–72
15. Hamann, L., Gogolla, M.: Endogenous Metamodeling Semantics for Structural UML 2 Concepts. In Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P.J., eds.: MoDELS. Volume 8107 of LNCS., Springer (2013) 488–504
16. Henderson-Sellers, B., Clark, T., Gonzalez-Perez, C.: On the Search for a Level-Agnostic Modelling Language. In Salinesi, C., Norrie, M.C., Pastor, O., eds.: CAiSE. Volume 7908 of LNCS., Springer (2013) 240–255
17. Igamberdiev, M., Grossmann, G., Stumptner, M.: An Implementation of Multi-Level Modelling in F-Logic. [5] 33–42