Towards Flexible Model Analysis and Constraint Development: A Small Demo Based on Large Real-Life Data

Matthias Sedlmeier and Martin Gogolla

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

Abstract. This contribution discusses the handling of a larger model on an abstract level employing the standardized modeling languages UML and OCL and an accompanying tool. We represent real-world data in form of a large object model and perform data analysis, verification, and validation tasks on the modeling level in order to obtain feedback about the original data. The tool allows to explore larger object diagrams interactively in a flexible way through a combination of visual and textual techniques. Furthermore, model invariants can be created in a versatile fashion by iteratively considering relevant object diagram fractions and evolving OCL expressions.
Keywords. Working with large models, Transformation and validation of large models, Visualization techniques for large models.

1 Introduction

Modeling and model management approaches have found their way into mainstream software production. Thus they are applied to large and complex systems, and approaches and tools have to deal with large models. In this contribution, we discuss an approach handling larger object models representing real-world data in a design tool originally elaborated for model validation and verification. The aim is to perform analysis, verification, and validation on the modeling level in terms of the languages UML [5] and OCL [8].

Working on the modeling level (in contrast to working on the code level or working with "production" systems as databases or programming languages) gives a high degree of abstraction through the available advanced modeling and validation options. UML and OCL have in contrast to traditional (relational query) languages the advantage of providing abstraction support for (a) different collection kinds (as sequences or ordered sets), (a) powerful operations (as closure or iterate), and (c) a mixture of textual and visual exploration mechanisms. Our approach gives interactive and direct feedback through the combination of textual and visual aspects, in particular for exploring model properties and model constraints. For large object models we support the exploration of queries and invariants that involve aggregation functions (that are of relevance in larger states

only) like the minimum or the average, as these functions are fully integrated into OCL.

Regarding larger object models, [4] identifies queries and expressions (e.g. in form of OCL) as an important field of activity. Our demo could further be part of an MDE tool benchmark focusing on queries [1]. Furthermore, structuring and slicing large class and objects models as in [7] is applicable in our context.

The rest of paper is structured as follows. Section 2 describes our process for obtaining our example model, in particular a large object model. Section 3 discusses the exploration of large object models by combining visual and textual techniques. Section 4 puts forwards an interactive process for constraint determination. Finally, Section 6 concludes our work and indicates further research.

2 Development Process for Class and Object Model

This section describes a real-life example for studying a large model. We wanted to obtain a model containing a UML class diagram and a large UML object diagram that can be handled interactively and flexible in *UML modeling and verification* tools. We employ the UML and OCL design tool USE (Uml-based Specification Environment) [3]. The tool supports model validation and verification for various UML diagram kinds, among them class and object diagrams, and OCL (Object Constraint Language) constraints in form of class invariants and operation contracts. Recently, the capabilities of USE have been improved in particular for handling large object diagrams (more efficient handling of object collections and evaluation of OCL expressions).

Our goal was not to compete w.r.t size with *production* tools as e.g. database systems. We aimed at an interactive development process exploring properties in the underlying data (in technical terms, in the employed object diagram) and exploring model constraints that have to be applied.

The German Federal Statistical Office (Destatis)[1] provides a municipal directory called GV100 of about 20.000 German administrative units, e.g., towns or villages, including about 60.000 structural connections based on a census in 2011. This directory can be officially obtained on the Destatis web portal as a monolithic column-oriented ASCII file together with record descriptions in natural language required to interpret the provided data[2].

The ASCII records contain information about federal states (German: Bundesland), districts (Bezirk), counties (Kreis), municipalities associations (Gemeindeverband) as well as municipalities (Gemeinde). Furthermore, the hierarchical connections between those units are given as well as the places of administration. We have complemented this data with additional geographical position information provided by the OpenGeoDB project[3] and obtain municipality records pro-

[1] https://www.destatis.de/DE/Startseite.html

[2] https://www.destatis.de/DE/ZahlenFakten/LaenderRegionen/Regionales/Gemeindeverzeichnis/Administrativ/Archiv/GV100ADQ/GV100AD3108.html

[3] http://opengeodb.org/wiki/OpenGeoDB

viding information about area, population, male population, female population, zip code as well as geographical information in form of longitude and latitude.

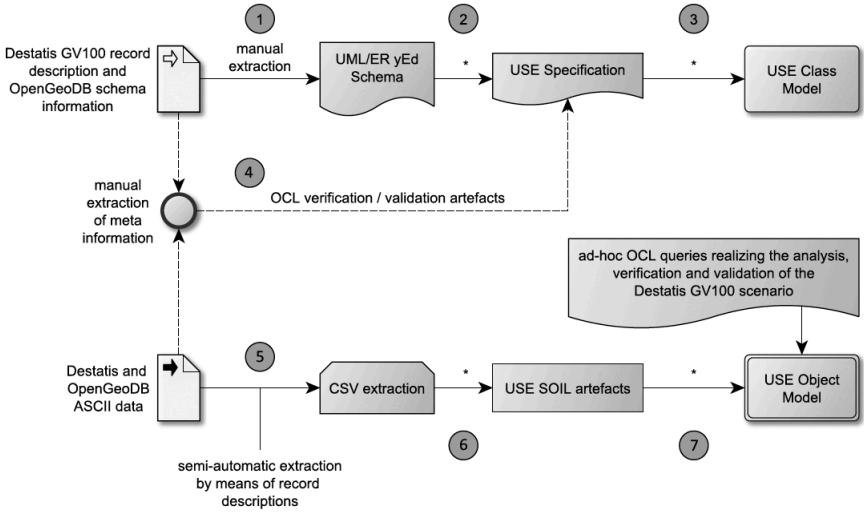

Fig. 1. Development process for UML class and object diagram

Figure 1 shows the process that we have applied in order to obtain an initial version of the class and object diagram. We started with the manual extraction of a data schema based on the Destatis GV100 record descriptions (1). As an intermediate representation, we have created this schema with the graph editor yEd in a UML/ER like modeling language (as in [6]). Moreover, we consider geographical information from the OpenGeoDB project.

In the next step, we perform an automatic transformation of the UML/ER schema into a valid USE specification (2) describing a USE class model (3). Figure 2 shows the central elements of the resulting UML class diagram. This model contains those classes and associations required for instantiating an object model. We have enriched the USE specification by OCL constraints and queries for verification and validation purposes (4), which are manually elaborated by analyzing the given record descriptions as well as the concrete ASCII records.

For the schema at hand, we semi-automatically extract the instance data as CSV files by means of Destatis record descriptions (5). The CSV data is then automatically transformed into USE data manipulation statements formulated in SOIL (6), which are interpreted by USE to instantiate the object model (7). SOIL ('Simple Ocl-like Imperative Language') is the USE "programming" language based on OCL. From this point on, we are able to interactively work in a flexible way with the class model including the object diagram employing the USE graphical interface, the command line interface and a combination of them in order to apply verification and validation tasks based on OCL constraints and queries.

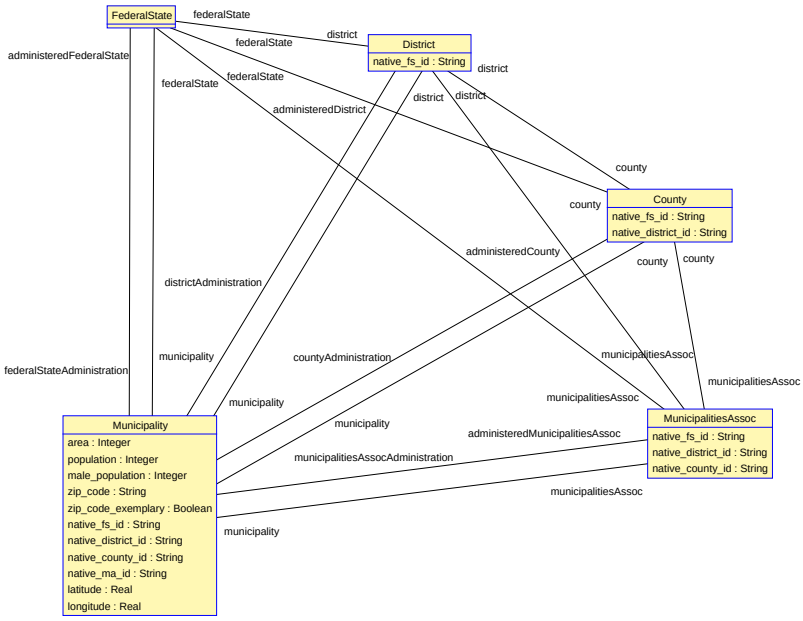

Fig. 2. Central elements of the UML class model for GV100

3 Exploring Object Models Visually and Textually

This section will explain the various interactive, but powerful options that USE offers to explore a large object diagram. The considered object diagram contains about 80.000 objects and links, as indicated in Fig. 3. Naturally, these objects do not fit on a single screen. USE allows to choose specific instances to create clearly arranged object model fractions. USE offers various object selection mechanisms [2], which effectively help to filter the desired subset. The techniques from [2] have been substantially extended in order to cope with object models showing the example's complexity. In USE, one can obtain fractions of the overall state fitting the developer's needs.

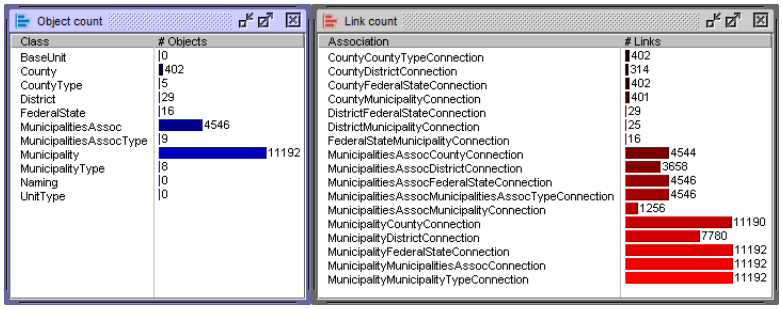

Fig. 3. Number of objects and links in an object model

First, it is possible to start with an empty object diagram to prevent long display times. We can then choose to show successively objects either by their type or by their properties.

Second, in addition to this simple filtering method, USE offers sophisticated object selection based on link path length, OCL query expressions and views. The latter method enables the developer to select objects specifically within a table view. The chosen set of objects can then be shown, hidden or cropped. A corresponding example is shown in Fig. 4, where a specific *FederalState* object (Schleswig-Holstein) and a *Municipality* object (Kiel) are manually selected by check boxes. Besides the objects, also all connecting links show up.

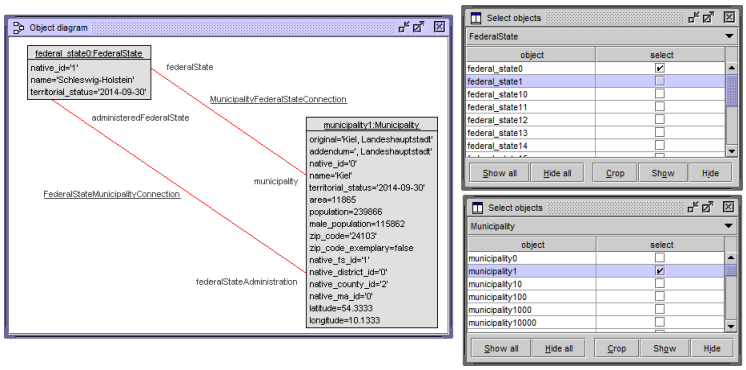

Fig. 4. Object selection based tables visually presented

Third, selection via link path length enables the designer to show only those objects, which are reachable from a given object over a specified number of link segments.

Fourth, the most flexible selection mechanism is provided via OCL query expressions, which allows the developer to show, hide or crop arbitrary sets of objects based on OCL query results being either single objects or object collections. Figure 5 shows an object model fraction containing only those federal states holding municipalities, which have a population above 100.000 and are located east of a fixed longitude and south of a fixed latitude. The example OCL expression only returns 3 objects, depending on the restrictness of the expression large object collections can be obtained.

Figure 6 illustrates a more detailed scenario, where some subordinated administrative units of the county *Merzig-Wadern* (where the Informatics meeting center "Dagstuhl" is located) are displayed. The example explores the administrative connections between all involved units (the county *Merzig-Wadern* belongs to the federal state *Saarland* and contains the municipality association *Wadern, Stadt*, which is at the same time also defined as a municipality; the county *Merzig-Wadern* also contains other municipalities like *Merzig*, *Mettlach* or *Losheim am See*).

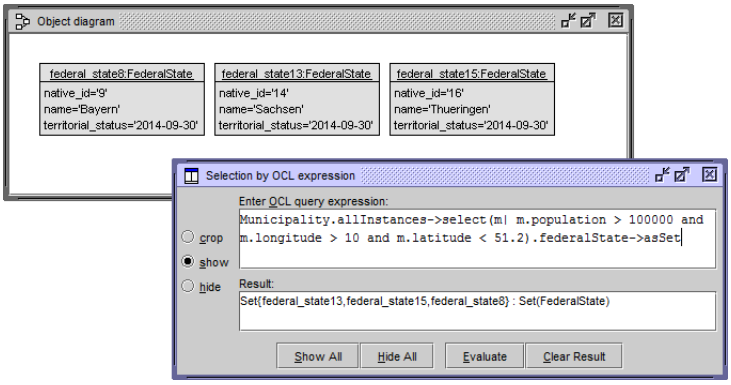

Fig. 5. Object selection by OCL expression visually represented

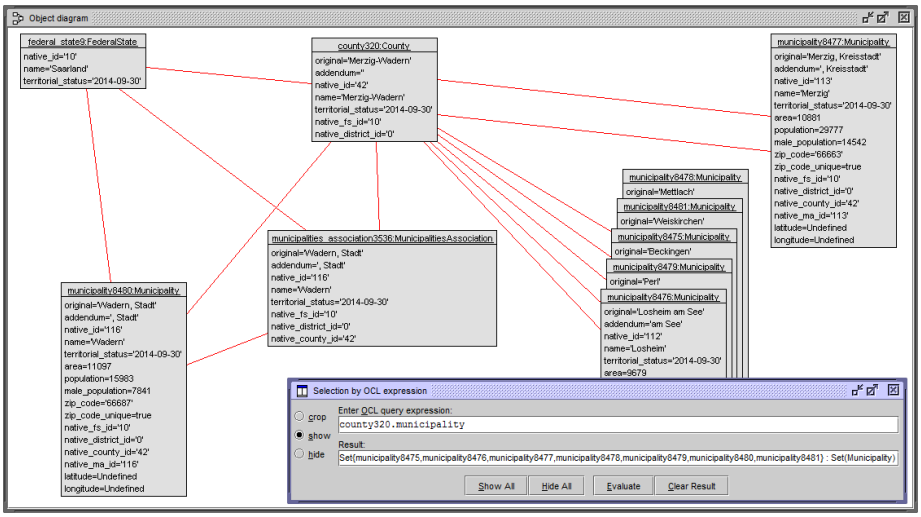

Fig. 6. Result of combining textual and visual object selection

Thus the model exploration options available in USE allow a combination of textual and visual techniques. The example object model display in Fig. 6 was created by evaluating multiple OCL expressions exploring the shown objects. USE provides the possibilities to construct object models stepwise with different OCL queries. This flexibility of switching between visual selection and evaluation of textual OCL expressions is, to the best of our knowledge, unique to USE.

4 Interactive Determination of Invariants

This section discusses how OCL model invariants can be developed interactively by considering the system state, formulating a hypothetical invariant, checking the assumed invariant against the state, and possibly revising the formulation of the invariant when the system state does not satisfy it and thus the invari-

ant cannot be validated. We follow an empirical approach, where we first make assumptions about rules, which should apply in the model application context. We therefore explore the schema information as well as the system state. Based on these assumptions we formulate an invariant, which we check by considering the system state in USE. If the invariant holds, we accept it as part of our USE model. If it does not hold, we analyze the result in order to determine, if our condition is invalid due to wrong premises or if the given object model (in the example determined by the Destatis information) is actually inconsistent. In the first case, we check if it is reasonable to adapt the invariant and start again. In the second case, we accept, that the underlying information (in the example the Destatis data) is possibly to a certain degree faulty and extend the USE specification with a modified invariant preventing the faulty information to go into the validation process (e.g., by weakening an invariant by adding an implication with a weakening premise).

In the example, the *longitude* and *latitude* attributes are not part of the original Destatis data but the information was taken from the OpenGeoDB project. This does not have any effect on the evaluation of the GV100 records. In fact, we would able to partly check the OpenGeoDB data, too.

4.1 Invariant *popGreaterThanZero*

We assume, that all municipalities have a population (pop) greater than zero:

```
context Municipality inv popGreaterThanZero:
  population > 0
```

We realize, that this invariant does not hold in the first inspection. USE allows us to check, which instances violate invariants either by the *Class Invariant View* or by the interactive shell `check -d` command. This is again an example where USE supports the developer through visual and textual techniques.

Through the analysis, we recognize, that all objects breaking our condition have the same municipality type assigned. This type classifies all assigned municipalities explicitly as uninhabited territory (German: 'gemeindefreies Gebiet, unbewohnt'). Based on this insight, we adapt our condition and accept it as part of our USE specification.

```
context Municipality inv popGreaterThanZero:
  (type.name <> 'gemeindefreies Gebiet, unbewohnt') implies
  (population > 0) and (male_population > 0)
```

This invariant holds in the system state, from which we can conclude, that the Destatis data is valid in this specific context.

4.2 Invariant *malePopLessPop*

Our premise is, that the male population must always be less than the overall population. This assumption is valid, because we explored the system state and validated, that

there are no "all-male municipalities" in Germany (the best pro-male ratio is 1:3.47 in *Freistatt, Niedersachsen* and the best pro-female ratio is 1:2.13 in *Hamm, Rheinland-Pfalz*). Computations like the best pro-male ratio can be formulated as OCL queries. In this case the ratio is determined by the following OCL expression:

```
Municipality.allInstances.collectNested(m|
  Sequence{m.federalState.name, m.name, m.population, m.male_population,
           m.population-m.male_population})
  ->collectNested(t|
    Sequence{t->at(1), t->at(2), t->at(3), t->at(4),t->at(5)
             t->at(4).oclAsType(Integer) / t->at(5).oclAsType(Integer)})
  ->asSequence()
  ->sortedBy(t|t->at(6).oclAsType(Real))
  ->select(t|t->at(3).oclAsType(Integer) > 0)->last
```

Taking the last invariant into account, we define:

```
context Municipality inv malePopLessPop:
  type.name <> 'gemeindefreies Gebiet, unbewohnt' implies
  population > male_population
```

The invariant is true in the considered system state. We accept it and conclude, that the Destatis data is correct in this context.

4.3 Invariant *zipCodeExemplary*

The *Municipality* class has a boolean attribute called `zip_code_exemplary`. This attribute indicates, that a single municipality can have multiple zip codes, but only one exemplary zip code is held in the attribute `zip_code`. However, we expect, that there must be municipalities having only one zip code, which we know from everyday life.

```
context Municipality inv notAllZipCodeExemplary:
  not Municipality.allInstances.forAll(zip_code_exemplary)
```

4.4 Invariants Based on Geographical Information

As we mentioned above, we enrich the Destatis data by geographical information. We try to assign the longitude and latitude value to each municipality by matching their names. This method is neither exact nor does it cover all instances. We also do not know, if the OpenGeoDB coordinates are always correct, since we did not compare them with third-party sources. However, we may assume, that all coordinates must be within the most northern, southern, eastern and western points of Germany. The following list explores the corresponding limits for all compass directions in the World Geodetic System 1984 format[4].

Based on this information, we postulate, that all longitude values must be in the stated interval. This should hold analogously for the latitude values. Given that, we derive 4

[4] http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf

direction	location	latitude/longitude
most northern	List, Sylt, Schleswig-Holstein	55.050000, 08.400000
most southern	Haldenwanger Eck, Oberstdorf, Bavaria	47.270108, 10.178319
most eastern	Deschka, Neieaue, Saxony	51.266667, 15.033333
most western	Isenbruch, Selfkant, North Rhine-Westphalia	51.050000, 05.866667

Table 1. Border points of Germany

similar invariants respecting the fact, that longitude and latitude values are not always present. All 4 invariants hold in the system state and we add them to the USE model.

```
context Municipality inv northLimit: -- analogously for south, east, west
  latitude <> null implies latitude <= 55.05
```

4.5 Derived Properties Employing Aggregate Functions

Instead of using longitude and latitude coordinates as constants, we can compute the bordering rectangle of a federal state with derived attributes as indicated in Fig. 7. The derived attributes for a federal state make use of the association with roles names federalState and municipality. This association indicates the municipalities that constitute the federal state. All municipalities of a federal state are considered there, and the most western (eastern, northern, southern) coordinates are computed. Another independent association is the one with role name federalStateAdministration. In a constraint one can now check that the coordinates of a federal state administration are consistent with the bordering rectangle of the federal state.

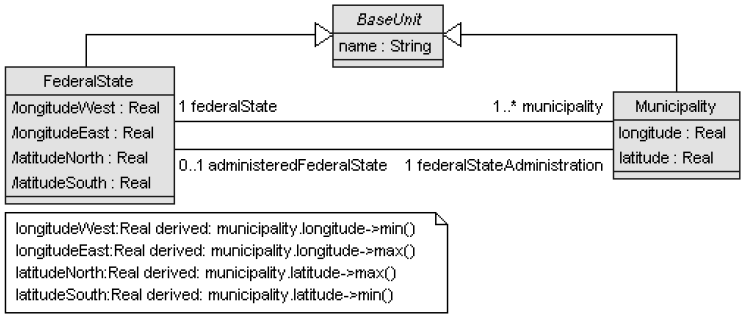

Fig. 7. Derived properties employing aggregate function in OCL

Another use of the bordering rectangle is a check about the plausibility of the bordering rectangles for different federal states. One can determine whether two federal states possess or do not possess common points (overlapping bordering rectangles). With that one can check that the actual present coordinates are such that, for example, the most northern state 'Schleswig-Holstein' and the most southern states 'Baden-Wuerttemberg' and 'Bavaria' do not have common points. We refrain from showing the detailed OCL expressions. But we emphasize that the derived attributes and the accompanying constraints only make sense in the presence of large object models as the attributes and constraints rely on aggregate function (here min() and max()) that apply in particular for large object collections.

5 Conclusion and Future Work

The practical experiences in our demo explores the capability of USE to handle larger object models with good performance. All discusssed expressions evaluate in magnitudes of at most few minutes. We have shown, how a large UML object model was employed for analyzing real-world data. The demo shows that analyzing data on the modeling level brings advantages w.r.t. to ease of formulating expressions due to the available abstraction mechanisms. We were able to browse and analyze the model and perform primary validation tasks. We used the provided textual and visual object exploration and selection mechanisms including OCL queries. In fact, our approach has an initial setup cost, but we obtain insight on a formal level with standardized modeling languages.

Further experiments will identify limitations with regard to the size of objects and links. Future work will also elaborate on fragmentation and slicing techniques for object models in order to handle representative slices of the original data. Another topic is the improvement of the visualization options for large object models, e.g., selection mechanisms by link kind (only association, binary and ternary associations, aggregation, composition). Naturally, more case studies and comparison with existing solutions must give feedback about the applicability of the proposal.

References

1. A. Benelallam, M. Tisi, I. Ráth, B. Izsó, and D. S. Kolovos. Towards an open set of real-world benchmarks for model queries and transformations. In D. S. Kolovos, D. D. Ruscio, N. D. Matragkas, J. de Lara, I. Ráth, and M. Tisi, editors, *Proc. 2nd WS Scalability in Model Driven Engineering BigMDE@STAF2014*, volume 1206 of *CEUR Workshop Proceedings*, pages 14–22. CEUR-WS.org, 2014.
2. M. Gogolla, L. Hamann, J. Xu, and J. Zhang. Exploring (Meta-)Model Snapshots by Combining Visual and Textual Techniques. In F. Gadducci and L. Mariani, editors, *Proc. Workshop Graph Transformation and Visual Modeling Techniques (GTVMT'2011)*. ECEASST, Electronic Communications, journal.ub.tu-berlin.de/eceasst/issue/view/53, 2011.
3. M. Gogolla and F. Hilken. Model Validation and Verification Options in a Contemporary UML and OCL Analysis Tool. In A. Oberweis and R. Reussner, editors, *Proc. Modellierung*, pages 203–218. GI, LNI 254, 2016.
4. D. S. Kolovos, L. M. Rose, N. Matragkas, R. F. Paige, E. Guerra, J. S. Cuadrado, J. D. Lara, I. Rath, D. Varro, M. Tisi, and J. Cabot. A research roadmap towards achieving scalability in model driven engineering. In *Proc. 1st WS Scalability in Model Driven Engineering (BigMDE 2013)*. ACM, 2013.
5. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language 2.0 Reference Manual*. Addison-Wesley, Reading, 2003.
6. M. Sedlmeier and M. Gogolla. Model Driven ActiveRecord with yEd. In T. Welzer, H. Jaakkola, B. Thalheim, Y. Kiyoki, and N. Yoshida, editors, *Proc. Int. 25th Int. Conf. Information Modelling and Knowledge Bases (EJC'2015)*, pages 65–76. IOS Press, Amsterdam, 2015.
7. D. Strüber, M. Selter, and G. Taentzer. Tool support for clustering large metamodels. In D. D. Ruscio, D. S. Kolovos, and N. Matragkas, editors, *Proc. Workshop Scalability in Model Driven Engineering (BigMDE 2013)*. ACM, 2013.
8. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 2003. 2nd Edition.