An Approach to Employ Modeling in a Traditional Computer Science Curriculum

or: Why Posing Essentials of the Object Constraint Language without Objects and Constraints?

Martin Gogolla

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

Abstract. This paper has two purposes: first, it discusses one approach that shows how modeling and in particular information systems modeling techniques are employed within a traditional computer science curriculum, and second, it explains and recapitulates essential concepts of a contemporary modeling language in a conventional style without stressing modern concepts. In the second part the paper focuses on the Object Constraint Language (OCL) without using objects and constraints. We use this as an example to explain how to summarize taught concepts in a good way. Experience on teaching modeling is constantly gained in three lectures on a basic, advanced, and specialized level, and roughly speaking, the three lectures introduce and concentrate on (1) syntax, (2) semantics, and (3) metamodeling of modeling techniques.

1 Introduction

The employment of modeling languages as the Unified Modeling Language (UML) [8, 6] and partly also the Object Constraint Language (OCL) [10, 7] during undergraduate and graduate Computer Science (CS) courses has been becoming a must nowadays. However, the position and importance of modeling languages in comparison to say, programming languages, widely differs.

This contribution is formulated from a perspective of a teacher that is involved in three main university courses: Database Basics (for the 2nd semester), Database Systems (for the 5th semester), and Design of Information Systems (for the 6th semester and higher). We have decided that each single course chooses a particular motto and concentrates teaching around that motto. The three mottos are: syntax, semantics, metamodeling.

Naturally, one cannot only teach the syntax of a language without explaining the evaluation, and thereby one will explain certain aspects of the semantics. However, the focus for the Database Basics course is on syntax, the core of the Database Systems course handles semantics, and in the center of the Design of Information Systems course is metamodeling. In our view, metamodeling is able

to combine syntactical and semantical aspects in a formal, but yet comprehensible way. One can build metamodels for the syntax and by employing that one can construct metamodels covering semantics. The categorization is not meant as a disjoint and exclusive categorization. So even elements from metamodeling may be shortly discussed during the Database Basics course, however in a very narrow setting, for example, when discussing the database catalogue.

When we are using the term 'traditional CS curriculum' we are referring to the fact that our curriculum was initiated by the end of the 1970 years and takes a view on software development that is 'code-centered' and that emphasizes the responsibilities that IT professionals have in respecting the society's demands. It has been reformed several times since then. However the reformations have been triggered, e.g., by practical questions around study organization, by changing profiles of study entrants or by the development of technology (e.g., the world wide web), but definitively not by considerations to give the curriculum a 'model-centered' character. The view on modeling, in particular, formal modeling, widely differs among members of our teaching team. Thus, the teachers supporting modeling try to inject as far as possible modeling principles and approaches disputing always with the more code-oriented representatives.

The work here has connections to related approaches. Our curriculum takes up central requirements from [2, 1] for the involvement of UML into our information system courses as far as the current situation in our department allows it. In all our courses we employ our UML and OCL design tool USE (Uml-based Specification Environment) [5] that gives direct feedback to course participants, in particular for the evaluation of OCL expressions in complex UML class diagrams. Our examples all obey the modeling principles [4] that we have developed earlier. Our approach emphasizes direct involvement of a formal tool with fast feedback steps and tries to minimize the gap between informal and formal modeling that was discussed in [11]. The approach [9] where new technologies as, e.g., smart phone apps are envisioned for course tests and that proposes an incorporation of social media into courses, seems to be good approach to reduce the distance between traditional and modeling-oriented approaches to teaching CS basics.

2 Teaching Syntax, Semantics, and Metamodels of Information System Models

As we have stated before, our experience relies on three major courses: (1) Database Basics (2nd semester, Bachelor degree, 2 lecture hours + 1 exercise hour, 4 ECTS, up to 250 participants), (2) Database Systems (5th semester, Bachelor degree, 4+2, 8 ECTS, up to 90 participants), (3) Design of Information Systems (6th semester and higher, Bachelor or Master degree, 4+2, 8 ECTS, up to 40 participants). The motto of each single course is (in the above order): syntax, semantics, metamodeling of information systems. These mottos are not meant to be exclusive or disjoint, but indicate the focal point of the respective course.

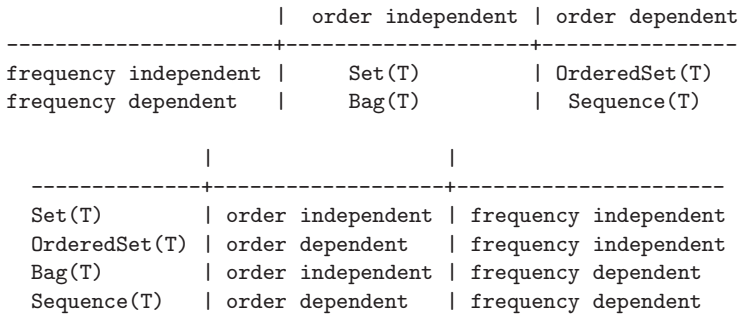

Fig. 1. OCL collection kind properties.

- In the Database Basics course the syntax of UML class diagrams (which is first taught in a different, accompanying software engineering course) is shortly recapitulated. Then the transformation into SQL Relational database schemata emphasizing the SQL options and the general need for database integrity is discussed. Integrity is handled in form of primary key, foreign key and check constraints. A large part of the course is devoted to SQL expressions as they are employed in SQL queries, data manipulation statements, view definitions, and check constraints. The students need to employ and practically apply the forwarded know-how in a two-semester Java development project. Thus the emphasis in the course is on practically applying SQL, and thus the focus is on the SQL syntax and an informal understanding of the constructs.
- In the Database Systems course an in-depth discussion of current and partly past data models is given: ER diagrams, UML class diagrams, object-oriented data models, the Relational data model (including Relational database theory with normal forms) and variations of the Relational model in form of non-first-normal-form approaches are discussed in a detailed way or are shortly sketched, as appropriate. For all data models the interpretation of schemata through database states, i.e., the semantics of schemata is put forward. The students need to thoroughly understand the differences between the data models, and thus the focus is on the schema semantics determined by the database states.
- The Design of Information Systems course concentrates on UML, OCL and our tool USE. UML class diagram concepts such as aggregation and composition or generalization constraints such as {overlapping, disjoint} and {incomplete, complete} are put forward in detail. The concepts are explained and their differences in terms of object diagrams are discussed with the help of OCL. The OCL syntax and the semantics of expressions and constraints is treated.

  In the last part the course treats metamodels. Therefore, a larger example for a UML and OCL model is introduced for the handling of the Entity-Relationship (ER) and Relational data model and their transformation on the schema and state level. A formal transformation between ER and the

Relational schemata respecting also semantic properties is put forward. This means that metamodels are used heavily, also transformation metamodels.

In a nutshell, the ER syntax metamodel introduces the classes ErSchema, Entity, Relationship, Relationship end, Attribute, and Datatype. The ER semantic metamodel treats for each syntactic class a semantic class (in the mentioned order): ErState, Instance, Link, Relationship end map, Attribute map, and Value.

The Relational syntax metamodel uses the classes Relational database schema, Relational schema, and again Attribute and Datatype. The Relational semantic metamodel interprets the syntactic classes through introducing classes for Relational database states, Tuples, and employing again Attribute maps and Values.

The connection between syntax and semantics is formally done in the metamodel with associations. The transformation metamodel concentrates on transformation correctness constraints that are attached to a Transformation class.

Apart from getting deep and fundamental insights into modeling languages as UML or OCL and into past and present data models, the students are prepared during this course for a thesis, either in the Bachelor or Master degree.

The three mottos, in the mentioned order (a) syntax, (b) semantics, (c) metamodeling, seem natural, at least to us. (a) In order to learn and apply a language you first have to know the basic rules how to arrange units to build sentences in the language. (b) You must know the fundamental syntactical rules. In order to pose meaningful language units you have to understand what you are saying or writing, and thus semantical considerations come into play. By learning the language to a larger extent you will enlarge your knowledge with respect to the available syntactical options and thus you have to get aware of more semantical colors and shades. (c) In the last step you may want to contemplate about the language as a whole with all its facets, let it be syntactical or semantical, and you may end up in thinking about and describing syntax and semantics in a uniform and systematic way.

Another is aspect is the question why we decided to concentrate on the mottos and to divide teaching the information system field into these aspects. Teaching on different levels, i.e., basic, advanced and specialized, requires to subdivide the contents into manageable units. One criterion could go along the languages (SQL Data Definition Language, SQL Data Manipulation Language, UML, OCL, etc.).

But this does not work in our curriculum, because basic knowledge about UML and SQL DDL is required already at the end of the first study year. Therefore we have chosen the three mottos which are orthogonal to the language dimension.

3 An Example for Content Summaries: OCL without Objects and Constraints

As one important ingredient of a course we want to discuss how to compile good summaries of taught content. Apart from mentioning all taught concepts in an easy understandable and appreciable form, we find it helpful to present a fresh view on the content. We try to avoid to only repeat already known material. A summary is intended as a help for students in preparing the examination, understanding the technical content and help to bring it up during the examination, let it be written or oral. The summary should have a clear structure helping to identify the central structural items. We try to obey this principle in all courses.

- For the Database Basics course, for example, a plain alphabetical list of SQL keywords is stated within an exercise asking the students to structure the list along common areas into which the keywords belong. The structure is asked to go along essential SQL query concepts using 'select-from-where-groupBy-having-orderBy', essential SQL logical connective concepts using 'and-or-not', essential SQL subquery concepts 'exists-in-any-all', essential SQL join concepts following join formulation 'natural-on-using' and join handling of null values 'inner-fullOuter-leftOuter-rightOuter', or SQL data manipulation commands 'insert-update-delete' (among other structure items).
- For the Database Systems course, for example, an ASCII-like syntax for basic and derived Relational algebra operations (product, union, difference, selection, projection, renaming; natural join, equi-join, theta join, intersection, division) is stated. The Relational algebra operations have been discussed before in set-theoretic, mathematical notation. In addition an implementation of the respective operations in SQL is pointed out.
- For the Design of Information Systems course, for example, essential concepts of OCL expressions are explained employing only values without using objects at all. This emphasizes in OCL the expression aspect and of course neglects the constraint aspect which is summarized at a different spot. Neglecting objects and constraints and concentrating on the expression character of OCL establishes a connection between OCL and general programming languages. It demonstrates that OCL expressions could be used inside a programming language, of course under the assumption that appropriate collection types are available in the programming language.
  Thereby we try to minimize the gap between programming and modeling. This also has the aim to bring the modeling-oriented way of teaching CS in a curriculum closer to the traditional, code-oriented way of teaching. It shows that on the technical basis both ways of teaching have transport similar and comparable technical content.
  The OCL summary in Figs. 2 and 3 is structured along a classification of OCL collection operations: constructors and 'destructors', basic boolean and integer query operations, advanced boolean query operations, advanced collection-valued query operations, complex query operation, coercions.

```
Constructors and `destructors'
- Set{7,8}, Bag{7,8,8}, Sequence{7,8,7}, OrderedSet{8,7,7}
- Set{}, Bag{}, Sequence{}, OrderedSet{}
- Set{7..9}, Bag{7..9}, Sequence{7..9}, OrderedSet{7..9}
- Set{}->including(8)->including(7), Bag{8,9,7,8,9}->excluding(9)

Basic boolean and integer query operations
- Set{7,8}=Set{8,7,8,7}, OrderedSet{7,8}<>OrderedSet{8,7}
  Set{7,8}<>Bag{7,8}, OrderedSet{7,8}<>Sequence{8,7}
- Set{7,8}->includes(8), Set{7,8}->excludes(9),
  Set{7,8}->includesAll(Set{8,8,7,7}), Set{7,8}->excludesAll(Set{6,9})
- Set{}->isEmpty(), Set{7,8}->notEmpty(), Set{8,8,7,7}->size()=2
  Set{7,8,7}->count(7), Bag{7,8,7}->count(7)
  Sequence{7,8,7}->count(7), OrderedSet{7,8,7}->count(7)

Advanced boolean query operations
- Set{7..9}->forAll(i|i>=0), Bag{7..9}->exists(i|i.mod(2)=0)
- Sequence{7..9}->one(i|i.mod(2)=0)
- OrderedSet{-9..-8}->including(8)->including(9)->isUnique(i|i*i)=false

Advanced collection-valued query operations
- Set{21..42}->select(i|i.mod(3)=0 and i.mod(7)=0)
- Bag{21..42}->reject(i|i.mod(2)=0 or i.mod(3)=0)
- Set{21..42}->any(i|i.mod(2)=1)
- Set{7,8,8}->union(Set{9,9,8}), Bag{7,8,8}->union(Bag{9,9,8})
  Sequence{7,8,8}->union(Sequence{9,9,8})
  OrderedSet{7,8,8}->union(OrderedSet{9,9,8})
- Set{-2..2}->collect(i|i*i), Set{-2..2}->collect(i|Sequence{i,i*i})
  Set{-2..2}->collectNested(i|Sequence{i,i*i})
- Set{-2..2}->collectNested(i|Sequence{i,i*i})->flatten()
- Set{-6,-5,-4,7,8,9}->sortedBy(i|i*i)
```

Fig. 2. Example OCL expressions showing OCL essentials avoiding objects (Part A).

```
Complex query operations
- Set{-2..2}->iterate(i:Integer;r:Set(Sequence(OclAny))=Set{}|
    r->including(Sequence{i,i*i,if i.mod(2)=0 then 'E' else 'O' endif}))
- Capitals: M[adrid], P[aris], A[msterdam], B[erlin], Z[urich], V[ienna]
  let TupleSet=
    Set{Tuple{s:'M',t:'P'},Tuple{s:'P',t:'A'},Tuple{s:'A',t:'B'},
        Tuple{s:'M',t:'Z'},Tuple{s:'Z',t:'V'},Tuple{t:'B',s:'V'}} in
  TupleSet->closure(T1|
    TupleSet->select(T2|T1.t=T2.s)->
      collect(T2|Tuple{s:T1.s,t:T2.t})->
        asSet())
                +-------------+
                |  select =   |
      Tuple{T1.s,T1.t}   Tuple{T2.s,T2.t}
                |          collect      |
                +-----------------------+

Coercions
- Sequence{8,7,8}->asSet()=Set{8,7}
- OrderedSet{8,7,8}->asBag()=Bag{8,7}
- Set{7,8}->asSequence()=Sequence{8,7}
  or Set{7,8}->asSequence()=Sequence{7,8}
- Bag{8,8,7,7}->asOrderedSet()=OrderedSet{7,8}
  or Bag{8,8,7,7}->asOrderedSet()=OrderedSet{8,7}
- Set{-2..2}->collect(i|i*i)->asSet()
```


Fig. 3. Example OCL expressions showing OCL essentials avoiding objects (Part B).

All expressions can be evaluated without having a class diagram available. Usually it is said that OCL can only be employed meaningfully within the context of a class diagram, and the concrete evaluation is then done within an object diagram. This summary demonstrates that this is not the case. The summary shows all central collection operations that are available for all collection kinds. It does not show the few collection operations that are defined only on particular collection kinds as for example `first` or `last` that can be defined only on `Sequence` and `OrderedSet`.

```
Seq{ Set{ Set{7,8}, Set{8,7}, Set{8,7,7}, Set{8,7,8} },
     Set{ Bag{7,8}, Bag{8,7}, Bag{8,7,7}, Bag{8,7,8} },
     Set{ Ord{7,8}, Ord{8,7}, Ord{8,7,7}, Ord{8,7,8} },
     Set{ Seq{7,8}, Seq{8,7}, Seq{8,7,7}, Seq{8,7,8} } }
=
Seq{ Set{ Set{7,8}                                    },
     Set{ Bag{7,8}, Bag{7,7,8}, Bag{7,8,8}            },
     Set{ Ord{7,8}, Ord{8,7}                          },
     Set{ Seq{7,8}, Seq{8,7}, Seq{8,7,7}, Seq{8,7,8}} }
```

Fig. 4. Illustration of OCL collection kind properties by a single OCL term.

All expressions can be evaluated in the UML and OCL tool USE [4] that is employed in our courses. The tool gives direct feedback to course participants for their exercises and is used in the classroom to demonstrate principle and enhanced aspects of example models.

The OCL collection kinds `Set(T)`, `Bag(T)`, `Seq(T)`, `Ord(T)` (we abbreviate `Sequence` by `Seq` and `OrderedSet` by `Ord`) together with their abstract superclass `Collection(T)` can be characterized independently from objects by two algebraic, abstract laws that express fundamental properties of collection kinds through stating equalities between constructor terms.

- `Set(T)` and `Bag(T)` are insertion order independent by obeying:

  ```
  C->including(E1)->including(E2) =
  C->including(E2)->including(E1)
  ```

  `Seq(T)` and `Ord(T)` are insertion order dependent.
- `Set(T)` and `Ord(T)` are insertion frequency independent by obeying:

  ```
  C->includes(E) implies
    C->including(E)=C
  ```

  `Seq(T)` and `Bag(T)` are insertion frequency dependent.
- A summary of the laws can be represented in tables as shown in Fig. 1.

As shown in Fig. 4 the facts can nicely be summarized by *four* set terms denoting *one* set, *four* bag terms denoting *three* bags, *four* ordered set terms denoting

two ordered sets, and *four* sequence terms denoting *four* sequences. Altogether the various terms are wrapped together into one single OCL term, so that the evaluation of this single term points out the central differences between the OCL collection kinds.

The term is of sort `Sequence(Set(Collection (Integer)))`. Basically, we want to explain with this term how the four different collection kinds in OCL behave differently, when the same collection elements are inserted resp. included in the same order. Altogether, we build four test cases:

1. Starting from the empty collection, for the first test case one first inserts `7` and then `8`. For the collection kind `Set` this can be represented as `Set{7,8}`.
2. Starting from the empty collection, for the second test case one inserts first `8` and afterwards `7`. For the collection kind `Set` this can be represented as `Set{8,7}`.
3. Starting from the empty collection, for the third test case one inserts first `8` and afterwards `7` and then again `7`. For the collection kind `Set` this can be represented as `Set{8,7,7}`.
4. Starting from the empty collection, for the fourth test case one inserts first `8` and afterwards `7` and then again `8`. For the collection kind `Set` this can be represented as `Set{8,7,8}`.

The four mentioned `Set(Integer)` literals all evaluate to the same set. Thus the four literals evaluate to *one* collection. If one uses `Bag` instead of `Set`, one obtains *three* different collections. If one uses `OrderedSet` instead of `Set`, one obtains *two* different collections. And if one uses `Sequence` instead of `Set`, one obtains *four* different collections. One uses the same construction rules for the different collection kinds, and ends up in different results in each test case. These test cases demonstrate the differences between the four collection kinds.

4 Conclusions

We have explained how information system modeling is integrated in different courses for various study years within a traditional CS curriculum. Structuring the courses along the mottos 'syntax, semantics, metamodels' works with reasonable success. However we would like to discuss how to improve the situation. One difficulty arising is that there are different views on the importance of particular, detailed modeling concepts in different CS areas. For example, to mention just a small, but important detail, association names are ignored widely in the programming language context (and are not taught with sufficient emphasis), whereas they are crucial concepts in the database area, because they must be used for naming tables in a Relational database.

Naturally also the different sublanguages of modeling languages as UML take different roles in different CS areas, for example, within the networking area sequence diagrams are the central formalism. Thus the idealized view 'Teaching

modeling ... is teaching abstraction' [3] is seen differently from the different CS areas. Another observation is that the link to theoretical CS, as it is present in the similarity between automata and UML statecharts, is often not emphasized enough and consequently not observed by students.

We have been injecting modeling topics around and within traditional CS content. We have been accepting and agreeing the kingdom view for CS (with kingdom names like Database Systems, Networking, or Programming Languages). What would it be like if one proceeds the other way round and looks at a curriculum from the central concepts point of view, i.e., if one would structure CS curricula around modeling principles and techniques, e.g., focus a complete curriculum around more abstract notions as 'model', 'metamodel', 'instantiation', 'abstraction', 'reduction', 'concern', 'refinement', 'syntax' or 'semantics'?

Acknowledgement

Thanks to the reviewers who have made many constructive remarks that have improved the paper and led to extensions. All deficiencies are of course due to the author.

References

1. ACM/AIS: IS 2010 Curriculum Guidelines for Undergraduate Degree Programs in Information Systems. ACM/AIS (2010) https://www.acm.org/education/curricula/IS%202010%20ACM% 20final.pdf.
2. ACM/IEEE: Computer Science Curricula 2013, Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. ACM/IEEE (2013) https://www.acm.org/education/CS2013-final-report.pdf.
3. Engels, G., Hausmann, J.H., Lohmann, M., Sauer, S.: Teaching UML is teaching software engineering is teaching abstraction. In Bruel, J.M., ed.: Satellite Events MODELS 2005, Int. Workshops, Doctoral Symposium, Educators Symposium, Springer, LNCS 3844. (2005) 306–319
4. 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, Berlin, LNCS 7949 (2013) 1–2 Invited talk.
5. Gogolla, M., Vallecillo, A.: On Explaining Modeling Principles with Modeling Examples: A Classification Catalog. In Chiorean, D., Combemale, B., eds.: Proc. 8th MODELS Educators' Symposium (EduSymp 2012), ACM Digital Library (2012) 28–31
6. OMG, ed.: Unified Modeling Language, Version 2.4.1. OMG (2011) OMG Document, `www.omg.org`.
7. OMG, ed.: Object Constraint Language, Version 2.3.1. OMG (2012) OMG Document, `www.omg.org`.
8. Rumbaugh, J., Jacobson, I., Booch, G.: The Unified Modeling Language 2.0 Reference Manual, 2nd Edition. Addison-Wesley, Reading (2005)

9. Scholz, M., Kaufmann, P., Seidl, M.: Making UML "hip": A first experience report on using modern teaching tools for object-oriented modelling. In Lethbridge, T.C., Stevens, P., eds.: Proc. MODELS Educators Symposium, ACM/IEEE 16th Int. Conf. MODELS. Volume 1134 of CEUR Workshop Proceedings. (2013)
10. Warmer, J., Kleppe, A.: The Object Constraint Language: Precise Modeling with UML. Addison-Wesley (2003) 2nd Edition.
11. Whittle, J., Bull, C.N., Lee, J., Kotonya, G.: Teaching in a software design studio: Implications for modeling education. In Demuth, B., Stikkolorum, D.R., eds.: Proc. MODELS Educators Symposium, ACM/IEEE 17th Int. Conf. MODELS. Volume 1346 of CEUR Workshop Proceedings. (2014) 12–21