Why (not) ORM?

Kresimir Fertalj, Nikica Hlupic, Lidia Rovan
Department of applied computing
University of Zagreb Faculty of Electrical Engineering and Computing
Unska 3, Zagreb 10000, CROATIA
kresimir.fertalj@fer.hr, nikica.hlupic@fer.hr, lidia.rovan@fer.hr

Abstract: *The two widely used approaches to data modelling, the application of Unified Modelling Language (UML) and Object Role Modelling (ORM) are compared. Since one of the purposes of modelling is to present and clarify the system design, the key comparison criteria were expressivity, i.e., clear presentation and easy perception of the stored information and substantiality, i.e., ability to store lots of information. A few characteristic situations were chosen that frequently arise in data modelling practice, and the examples clearly show how the same information can be very differently perceptive in ORM and UML models. Thus, the paper can help in selection of the modelling framework for a particular project.*

Keywords: software modelling, data modelling, comparative analysis

1 Introduction

System and database modelling is quite an old discipline, dating back to the very beginning of the modern computer science. Even the first software projects required some planning and design before coding. In 1976 Peter Chen introduced entity/relationship modelling [1]. After that, modelling got a widely accepted framework. Rapid future growth of software business has followed. Object-oriented programming strongly motivated further development of modelling. At present there exist several commonly used system and data modelling techniques [2]. Regardless of their notational conventions, all of them have the same purpose. They specify the data that a system needs to collect, they describe relationships and the usage of these data. Having the same purpose implies certain similarities. The models that were created by majority of the commonly used techniques are automatically mutually convertible. Convertibility and the fact that the most noticeable differences among the techniques are primarily of the graphical nature, can make impression that the choice of the technique is of minor importance. However, the purpose of modelling is also to present and clarify the system design. We mainly expect from the models *expressivity* (clear presentation and easy perception of the stored information) and *substantiality* (ability to store plenty of information). Applying these two properties as the key criteria for evaluation reveals many subtle, but significant differences among the compared techniques.

Most of references that can be found in the literature are of a wide scope, comparing several modelling techniques simultaneously [3] and usually with respect to many various properties. This paper takes a different approach, based on *expressivity* and *substantiality,* avoiding too much technical details. In addition, we shall restrict ourselves to comparison of ORM and UML as the two widely used representatives of the two opposed modelling concepts, i.e., UML as mainly software modelling [4] and ORM as data modelling [6][7] (in fact, UML itself is not well suited for database design, so we use the UML extension for database design). Furthermore, instead of an extensive comparison, which can hardly yield precise and unambiguous answers, we compare these two techniques under a few characteristic circumstances that frequently arise in practice. The examples will clearly show how the information provided with ORM and UML models can be very different, that is, very differently "perceptive".

2 ORM and UML expressivity and substantiality

2.1 UML and ORM graphic

Let us start with Figure 1, which contains an extract of a university information system presented in UML (Figure 1-a) and in ORM (Figure 1-b) models.

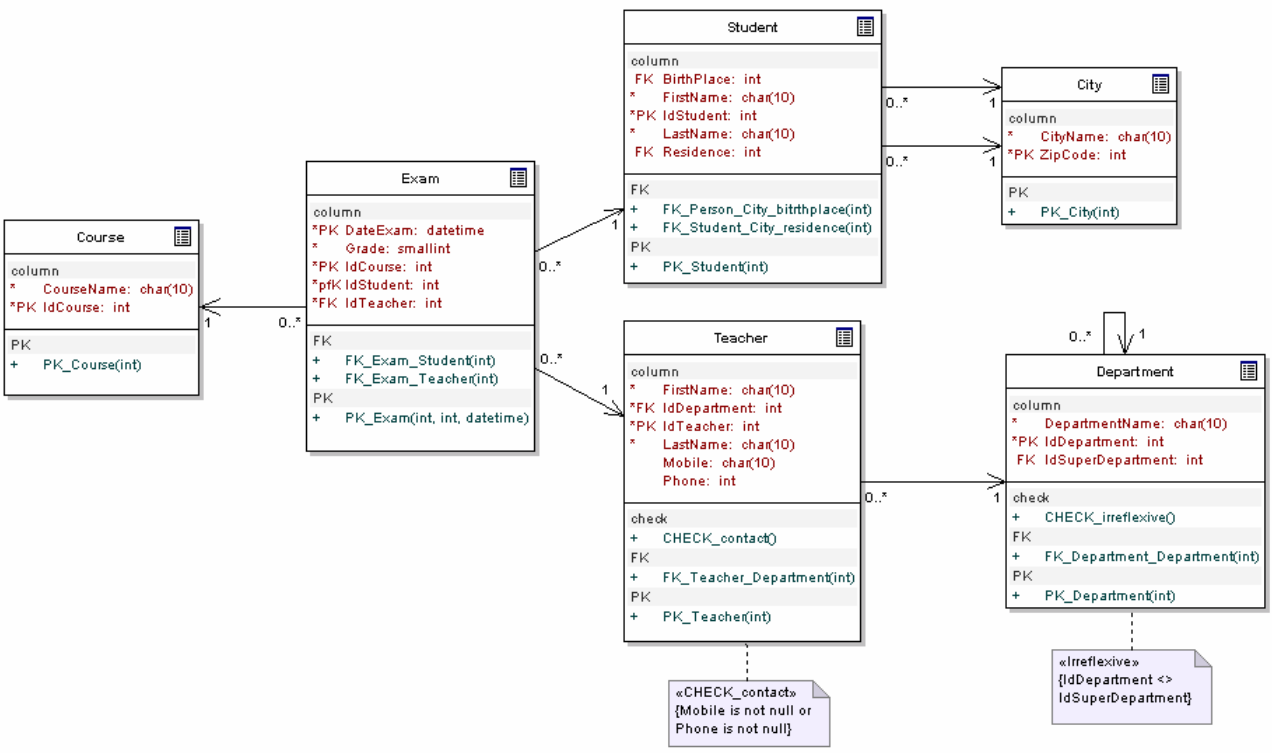

Figure 1-a: UML model of a part of university IS (counterpart to ORM model in Figure 1-b)

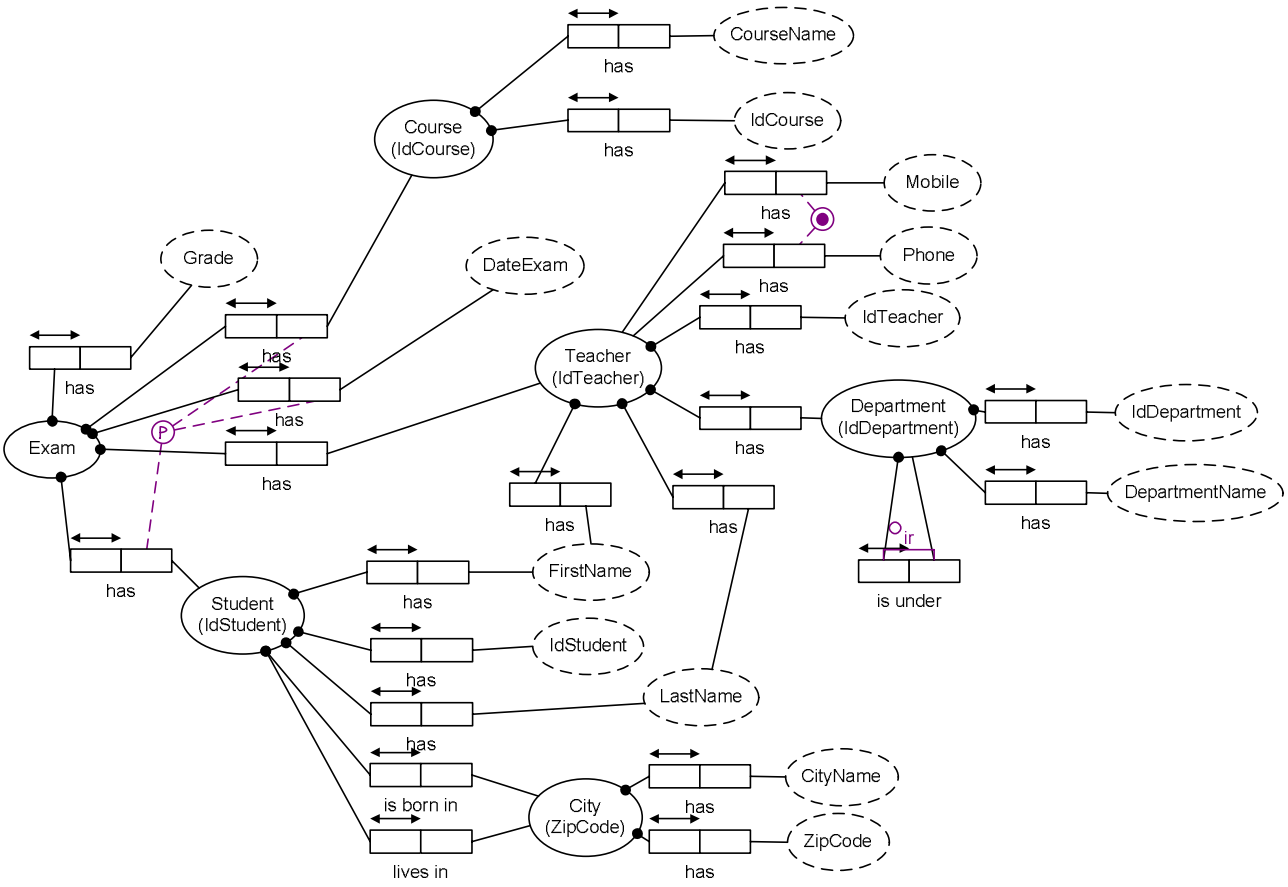

Figure 1-b: ORM model of a part of university IS (counterpart to UML model in Figure 1-a)

Both parts of the figure represent the same information system, in which the "central" entities are Teacher and Student. In this simplified example, Teachers "come from" Departments and Students from Cities. There is also Exam table, whose records can be stored and are unique only when all three "participants" are known, that is, Course, Teacher and Student.

Looking at Figure 1, it is obvious that UML representation is very neat, concise and readily understandable, while the ORM model looks like a mess at first glance, especially to someone not familiar with it. Our main complaint is related to the inadmissible graphical similarity of symbols for entities and their attributes. They are geometrically the same (ellipses), just drawn with different types of line (attributes are dashed), so one has to look the diagram very carefully to be able to trace relations among the entities. In contrast, UML encloses the attributes into the entities (tables), providing a clear system overview. Not surprisingly, for complex models, ORM notation usually produces huge graphs, because all the attributes of each entity have to be denoted with their own symbols and represented by separate graphical objects. Moreover, every attribute has to be connected with the corresponding entity by a predicate. It is another graphical symbol providing information about the relationship between the attribute and the entity. The result is that a single UML graphical object (table) maps to a densely "populated" ORM figure. Even small systems comprising just of a few tables can become hardly readable in ORM notation. Since both models contain the same information, their overall substantiality is equal, but we may say that UML expressivity is much better. As a consequence, ORM is certainly not the preferable technique for high-level design or management-level business discussions. The ORM mess of attributes and relations hides the objects of the primary interest and deliberations. For presentation or documentation of the general system design in a quickly and easily understandable notation, we consider UML as a much better choice.

The "untidiness" of ORM diagrams reduces their expressivity and it can sometimes produce ambiguities, even for experienced readers. Observe for example, Student and City entities in Figure 1. Every student has birthplace and city of residence, which, of course, do not have to be the same. UML diagram clearly shows two tables and two oriented arrows between them, indicating parallel relationships (double connection), with two foreign keys in Student table. Even those who do not understand the meaning of UML notation can immediately notice that there is something specific between these two entities, when compared to the rest of them. There are two clearly visible arrows, and everywhere else, there is only one. In ORM diagram it is hard even to find a particular relationship, let alone to notice its "speciality". All the relations look alike and there are too many of them for any to be singled out as especially noticeable. Furthermore, it seems in Figure 1 that both lines from City end up in the same point on Student. The Figure 2 shows the reality (the point denotes mandatory attributes). In complex diagrams with many attributes this might become a problem. There is only one solution – a manual rearrangement of the diagram.

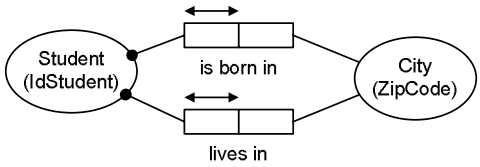

Figure 2: Clarified ORM presentation of relationship between Student and City entities

As counterexample and a rare exception, which shows that untidiness of ORM diagrams does not unconditionally mean worse expressivity, we can mention the primary key symbol (encircled P) in Figure 1. ORM draws it as a "stamp" over the model. In complex diagrams these lines will recklessly cross over other items. This looks untidy, strengthening the mess impression even more. However, because of its inconvenient placement, this symbol is very noticeable and it clearly points to the attributes that form the primary key. UML simply denotes the corresponding attributes with "PK", avoiding the need for new graphics in the figure, but making these attributes not at all "special" and nothing better visible than the others. Information is complete in both models, so neither has better ability to store primary keys, that is, substantiality is equal. Nonetheless, expressivity can be very different depending on complexity of the model, this time giving ORM a slight, but rather unimportant advantage.

As the last, but serious, complaint to ORM notation we shall once again take note of its modelling of relationships. We have already mentioned that there are too many of them and they are all the same, but "all the same" just meant that all relationships among entities look the same. Unfortunately, this is true for the

relations among entities and their attributes as well. As a consequence, looking at ORM diagram we are not able to distinguish connections of entities with their attributes from relationships among entities, at least not quickly and easily. This is very bad, since what is important in both, the system and database design, is to model and trace relationships among the entities, not relations among entities and their attributes. Moreover, the ORM model does not give us a single clue about what attributes and how participate in a relationship. What we see is that a relationship exists and that is all. This is not just bad; it is awful.

Notation for optional attributes or cardinality in entity relationships in ORM is also an example of inconvenience. This information is provided by the arrows above predicates, with the position of arrows denoting one to one, one to many etc. types of relationships. Figure 3 shows all the combinations. Teachers have reserved parking places, every teacher only one, so each parking place can be associated with only one teacher. Bank accounts can belong to only one teacher, but every teacher can have more than one account. Teachers can lecture several courses and every course can have several lecturers. Finally, every teacher lives in exactly one city, but many teachers can (and probably do) live in the same one.

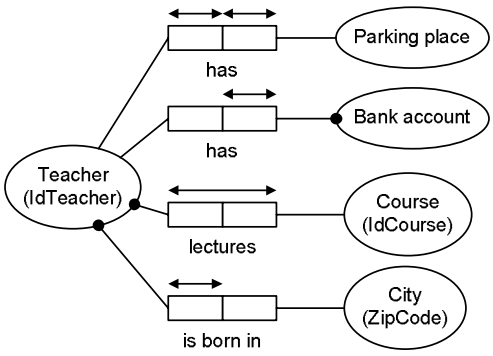

Figure 3: ORM notation for cardinality

While ORM notation is acceptable in technical sense, because we can model any cardinality, there are too many symbols for relations, and consequently too many arrows in the diagrams for such notation to be easy readable. On the other hand, UML indicates the direction of references by oriented arrows, cardinality and optionality of the attributes by numbers attached to the arrows (see Figure 1) and denotes the attributes that participate in relationship as primary or foreign keys. No matter how clear or unclear may it be, it is still incomparably better than what ORM offers. Certainly, it is easier for people to read the numbers attached to only a few arrows than to carry about the position of a few arrows among the tens of other identical ones. Thus, regarding modelling of relationships, UML is the absolute winner, being much more expressive.

2.2 UML and ORM "obligingness" and automation

Taking technical details into the consideration, especially for database design at lower design levels when detailed description of particular parts of the system is required, things become more complicated and less obvious and exclusive. UML will mostly produce neater graphs and provide better expressivity, but in some scenarios it may not completely "understand" the user's thoughts or might not be able to automate the work as much as ORM does. In Figure 1 we have two such examples, for convenience shown in Figure 4 and Figure 5.

Let us first consider the case in Figure 4, which shows disjunctive constraint on Teacher's "Mobile" and "Phone" attributes. ORM displays it with a special symbol, again placed somewhat untidy, but well noticeable (Figure 4-c). UML displays constraints as class methods, so it just adds another line to the lower part of its table, making the constraint neither more, nor less noticeable than other items. The developer has the option to attach a note-box to the table (Figure 4-a and Figure 1), which emphasises the constraint very much, but it requires quite a lot of space in the figure. A variant is to add a comment or some other text directly into the table, immediately under the name of the constraint (Figure 4-b). This requires less space, but it is therefore less noticeable. All in all, regarding expressivity of constraints, ORM and UML are quite equal, assuming that the reader of the diagrams knows the meaning of ORM special symbols.

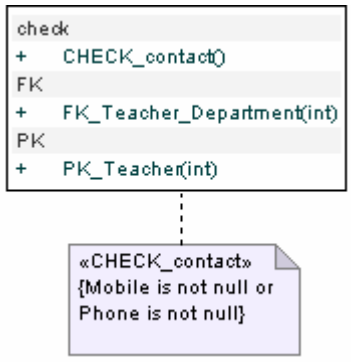

Figure 4-a: UML representation of disjunctive constraint, first variant

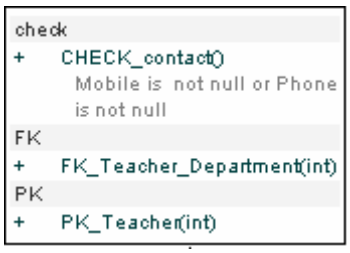

Figure 4-b: UML representation of disjunctive constraint, second variant

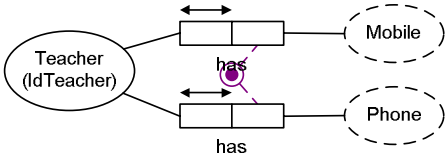

Figure 4-c: ORM representation of disjunctive constraint

Technically, a disjunctive constraint means that we demand at least one of the attributes not to be NULL. Such constraint requires a background code to check the entered values each time a new Teacher is added to the database. ORM offers several types of constraints and, depending on the developer's selection of the type, automatically generates the required SQL code for checking the constraint. The code generated for our example is

```
alter table "Teacher" add constraint Teacher_MR check ( ("Phone" is not null) or ("Mobile" is not null) ).
```

UML does not automate this task. The developer has to write the code manually. We may say that ORM is a more advanced and "developer friendly" tool in such occasions.

The other example is an irreflexive type of ring-constraint in Figure 5. This constraint means that the object (entity) cannot be referenced by itself, i.e., it cannot contain itself. In our example, a Department can have other (sub)departments, but cannot comprise itself. ORM, as always, provides special (untidy) symbol for such constraints (Figure 5-b), while UML adds a new "class method" (named "CHECK" in our example). However, this time UML also adds a backwards oriented arrow (Figure 5-a), which is a very noticeable and a very neat symbol, so we give to UML complete graphical advantage for ring-constraints.

Yet again, ORM automatically generates the appropriate code for such constraint, in our example

```
alter table "Department" add constraint Department_ring check ( "Department1" <> "Department" ).
```

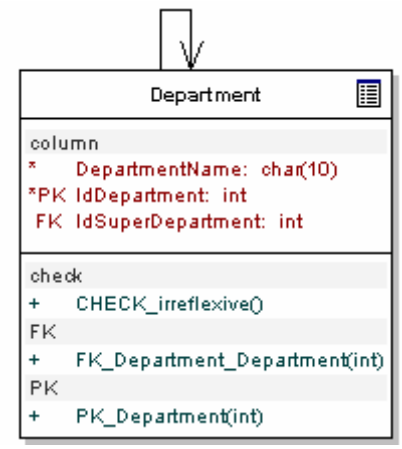

Figure 5-a: UML representation of ring-constraint

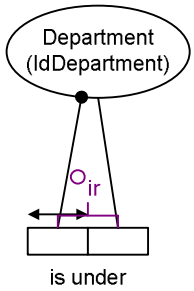

Figure 5-b: ORM representation of ring-constraint

Table 1: Final observations about UML and ORM

	UML	ORM
clarity of system design	very good, easily readable	very bad, crowded diagram
traceability of relationships	good	bad
notation of cardinality and optionality	clear and obvious	special notation, not obvious
special relationships	easily noticeable	not enough different from "normal" ones
constraints – notation	good	good
constraints – automation of work	sufficient	advanced
hierarchy, inheritance	quite well, clear	sufficient, not obvious
domain integrity	yes	no
indicated visibility of attributes	yes	no

Although ORM is sometimes more "obliging" than UML and automates certain steps of modelling, we do not regard it as a significant advantage, because it refers mostly to minor tasks, like adding just a few lines of code. Since there is almost no information that cannot somehow be stored in both models, substantiality is equal, but sometimes not equally "accessible". Neither the expressivity was notably different in the presented situations. It appeared sometimes

better in ORM, and sometimes in UML. Thus, regarding the technical properties, it remains a matter of personal preference which technique to use. Before making final conclusions, here we provide Table 1, which specifies most important observations that we made in previous considerations.

3 Conclusion

ORM was among the first [5], and it was one of the best modelling techniques for a long time. Nonetheless, according to all the presented comparisons, we believe that it is time for the ORM retirement. It is not bad, but in the world of object-oriented programming and conceptual system design, much younger UML is simply better.

The term "better" does not have purely technical meaning. There are not many things UML can and ORM cannot do, but UML does the same things more conveniently. We have shown in the paper that UML expressivity is much better. People can percept the information stored in UML diagrams quickly and easily, even without being familiar with UML notation. Thus, for conceptual design and high-level deliberations, ORM is not a worthy competitor to UML at all. On the other hand, substantiality showed to be approximately the same, so we can put many details into the both, to UML and to ORM models. However, the question arises what is the purpose of the model. If we have to embed many details into the model, then the model is certainly not intended to management-level deliberations or conceptual design. If it is not intended to high-level design, then it is intended to developers and programmers, and in that case the expressivity is not so important, because developers can read models of any complexity and understand all the symbols and the technical details embedded in graphs. Thus, at low design level expressivity is not important any more, and we have already found that substantiality is equal. What is important then? The only remaining criterion is usability of the model for further system development. This means how much of the previous results can be reused later and how much of the remained job can be automated. Again UML has the technical advantage, because it is much "younger" and as such comprises more modern concepts than ORM. For example, if we have properly prepared class diagrams, UML can generate much of the code automatically. As we can see, UML wins in all scenarios, and there is no doubt that it is the recommendable technique at present and certainly in the near future.

In this paper we tried to shed light on the two most popular modelling techniques from somewhat unusual viewpoint. The key criteria were expressivity and substantiality, expressivity being a very subjective impression. In other words, there were things in this comparison that are not exactly "measurable". The observations and most of conclusions were derived from our personal experience, so all conclusions we stated have to be adequately interpreted. In spite of probable subjectivity, we performed this analysis in this way intentionally. We tried to take into account the human psyche, that is, expectations of and perception from the models created by UML and ORM. Although this cannot be mathematically exact, we believe that results of such experiments are the only real indicators of the usability of a certain tool or technique. Therefore, we pay a great attention to such analyses and experiments, and hope to motivate the others to similar research.

References:

[1] Chen PP. The Entity-Relationship Model — Toward a Unified View of Data, ACM Trans. on Database Systems, March 1976; 1(1): 9-36.

[2] Wieringa R. A Survey of Structured and Object-Oriented Software Specification Methods and Techniques, ACM Computing Surveys, December 1998; 30(4).

[3] Hay DC. A Comparison of Data Modeling Techniques, Essential Strategies, Inc; 1999.

[4] Object Management Group. UML 2.0, The Current Official Version, http://www.uml.org/#UML2.0.

[5] Falkenberg ED. Concepts for Modelling Information. In: Nijssen GM. (ed.). Modelling in Database Management Systems. Amsterdam: North-Holland Publishing; 1976.

[6] Halpin, T. Object Role Modeling (ORM/NIAM), In: Bernus P., Mertins K. & Schmidt G, editors. Handbook on Architectures of Information Systems, Springer-Verlag, Berlin; 1998. p. 81-101.

[7] Halpin T. A Review of Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. The Morgan Kaufmann Series in Data Management Systems; April 2001.

[8] Kristensen BB. Object Oriented Modelling with Roles, In: Proc. of the 2nd Intl. Conference on Object Oriented Information Systems, OOIS '95; 1995, Dec. 18-20, Dublin, Ireland. London , UK: Springer Verlag Limited; 1995, p. 57-71.

[9] Navathe SB. Evolution of Data Modeling for Databases, September 1992; 35(9): 112-123.