

Normalization Rules of the Object-Oriented Data Model

No Author Given

No Institute Given

Abstract. There are only very few approaches to normalizing object-oriented data. In this paper we present an approach to normalization of the object-oriented conceptual model based on UML class diagrams. First part of the paper describes the current status in the area of formal methods used for object-oriented data modeling. Second part presents four normalization rules, which are based on own experience and modified Ambler-Beck's approach. These normalization rules are introduced on an example. Our method has been used in education at several universities. It has been and is also used for database design in software development projects, which we carried out. Recently, development of the CASE tool based on this approach has been started.

Keywords. Data normalization, object-oriented data model (ODM), first object-oriented normal form (1ONF), second object-oriented normal form (2ONF), third object-oriented normal form (3ONF), fourth object-oriented normal form (4ONF).

1 Introduction

Nowadays many various kinds of object-oriented software applications are used practically. [7,10] We have the long-term experience with Gemstone database and Smalltalk programming, for example. Although there already are many theoretical works, which individually demonstrate suitability of non-relational object-oriented data model, only the procedures based on experience with imperative object-oriented programming languages are used in the area of analysis and design of data structures. However some techniques like behavioral design patterns or object library components, which are optimal for algorithms in software application, can fundamentally complicate their effective data processing. As a consequence of this situation, we can see wrong usage of relationships and hierarchies among objects, breakneck tricks in code, etc. The problem of these applications is not that they do not work. Unfortunately really monstrous constructions work thanks to modern programming libraries and development environments. This is why the discussion with designers about the need to rebuild their software is very hard.

Moreover, relational design techniques as data normalization, data decomposition and data synthesis cannot be easily used in object-oriented data structures. This is why various proposals of object-specific normalization techniques

appeared in the world of software developer's community. Unfortunately no generally accepted and widely used technique or method for object-oriented data design has been introduced so far. Our solution to this problem is adapting the Ambler-Beck approach. [1,2,4] It has been developed as a part of agile programming techniques. Our contribution is in modification of this approach towards specific data structures in object-oriented models.

Therefore we decided to discuss the formal techniques of object-oriented design. A data structure is the foundation of almost all software applications. The object-oriented technology has the status of the mainstream. But, many myths exist in the community of object-oriented software vendors and developers. For example very popular is the myth about no need for any normalization, no need for any other formal technique, and about easiness of programming etc.

2 The issue of different software technologies

2.1 MDA

MDA is an abbreviation for Model Driven Architecture. MDA defines an approach that separates a specification of business system description (CIM – Computation Independent Model) from its computer implementation specification (PIM – Platform Independent Model); and this computer specification from the final solution on a concrete technological platform (PSM – Platform Specific Model). Each specification represents an individual viewpoint of the same problem. According to MDA, there is a mutual relationship between these three views, and the models should transform from one to another when a system is created. MDA is created and maintained by the Object Management Consortium. [11]

2.2 Object-oriented programming

The object-oriented approach has its origins in the researching of operating systems, graphic user interfaces, and particularly in programming languages, that took place in the 1970s. It differs from other software engineering approaches by incorporating non-traditional ways of thinking into the field of computing sciences. We look at systems by abstracting the real world in the same way as in ontological, philosophical streams. The basic element is an object that describes data structures and their behavior. In most other modeling approaches, data and behavior are described separately, and, to a certain extent, independently. OOP is explained in many books, but we think that this one [7] written by OOP pioneers belongs to the best.

The OOP can be regarded as one implementation option of PSM from more possible. The interesting question is the position of PIM. In ideal case, this model should be independent on the following PSM. However, this does not happen in practice. Either the object-oriented data model derived from the UML or older Entity-Relation data model, which is closer to the relational database technology,

is usually used for conceptual modeling on the level of abstraction correspondent to PIM.

If we try to figure out how the independent conceptual data model for PIM should really look like, we will find out, that we need to use following modeling concepts:

1. Entity.
2. Link between entities - however we need to distinguish between:
 - (a) IS-A relationship, i.e. taxonomy or inheritance,
 - (b) ASSOCIATED-TO relationship, i.e. link creating tuples of entities, and
 - (c) HAS-A relationship, i.e. link describing hierarchic structures or data compositions.

Detailed overview of various approaches can be found in the Table 1. It is obvious that on the conceptual modeling level, the relational data model and existing object-oriented data model are incompatible. That is why we presume that formal techniques known the relational database field are not suitable for object-oriented data modeling and vice versa.

A concept of object identity is the next problem of simple adoption of relational technique for the object-oriented data model. In RDM, the identity of record is created by a value of chosen attributes (primary keys). In object data model identity of object is based on addresses into virtual memory and is independent on any value changes.

2.3 Object-oriented databases

Database systems are based on various data models, e.g. *network* (and its subspecies hierarchical data model), *relational*, *object-relational* and *object-oriented* data model. [10,5]

Nowadays relational database model dominates. But recent practice shows that object databases are able to compete with relational databases. Object databases are based on two substantially different data models:

1. *Object-relational* (or hybrid) data model (ORDM) introduces an evolutionary trend. It concerns on addition of the original relational data model with the support of some structures and operations known from programming languages. Most of the big producers of relational database systems (e.g. Oracle) chose this alternative. Object relational data model stays principally on the same relational data model, but with extended functionality.
2. *Object-oriented* data model (ODM) introduces a new revolutionary trend of development. It concerns new data model, which is not built as an extension of relational data model at all.

The impedance problem with storing and retrieving of object-oriented data in relational and also in object-relational databases was the main reason for ODM.

feature	model	comment
R	Entity-Relational	This is the traditional RDBMS model based on Chen. <i>Here is the area of the relational normalization.</i>
C	Network	This is the model of the network databases (IDMS).
I	???	This conceptual model does not exist. Or does anywhere?
RC	Hybrid Network and Entity-Relational	This is the RDBMS model combined with data containers (e.g. NF^2 = non-first normal form databases).
RI	Extended Entity-Relational	This is the RDBMS model extended by the inheritance (e.g. IDEF1X and some object-relational DBMS).
CI	OOP Data Model	This is the data model of the recent object-oriented programming languages (e.g. Java, Smalltalk, C#, ...) and many programming-language-based OODBMS. <i>Here is the area of our approach.</i>
RCI	The universal conceptual model	This data model includes all conceptual features and reflects the proposed ODMG 3.0 and UML 2.0 standards, but is not yet directly implemented in recent object-oriented programming languages. <i>Do we need any universal normalization technique?</i>

legend

R - presence of the association relationship (i.e. RELATED-TO).

C - presence of the composition relationship (i.e. HAS-A).

I - presence of the inheritance relationship (i.e. IS-A).

Table 1. Possible approaches of the conceptual data modeling.

[10] This is the reason, why the construction of new database models, which would be able to work with objects better, has risen.

ODM and RDM differ distinctively from each other. In RDM, tables are the only possible form of logical data representation and their physical storing as well. On the other hand, ODM is similar to network databases, as we knew them in IDMS systems. The ODM can be interpreted as the renaissance of network data model. In a very simple way, it can be described by the following equation:

$$\textit{network data model} + \textit{object oriented paradigm} = \textit{ODM}$$

It is reasonable to expect that the role of object databases will grow in the near future, because there are many applications, where object-oriented database shows its advantages. Common attribute of these applications is large amount of complex data structures and their variability during their lifetime. Those systems can work with more than hundred or thousand various mutually composed and changing data types. Moreover, the queries over these structures require common polymorphism and abstraction. In those systems, for example, we need to write down the queries over sets containing elements of various types. At the same time we expect that while adding or updating data types it will not be required to change already written queries and related data structures. Good example of those systems are data warehouses. Those systems are characteristic not only for company management systems, but also for various governance evidence systems, hospital information systems and information systems containing ecological information, agricultural information, historiographical information as well, decision support in marketing and finance [7,9,10].

On the other hand it is necessary to note that relational database works very well in area, where database structure is stable. This means that new data types are not added during lifetime of such system. Moreover, relational databases traditionally achieve very good performance if the database consists of large amount but simply structured records.

3 Miscellaneous approaches of object-oriented normalization

Some various papers aroused since 1980s (for example [15]). First papers applied to the enlargement of relational techniques, but we can meet the papers specialized to object-oriented data structures in recent last years. There are several research groups in the world, which are interested in object databases. The international organization ODMG – Object Database Management Group supports publications and conferences on this topic. [5,14]

3.1 Nootenboom's OONF

According to Hank Nootenboom the first three relational normal forms are universally valid for the object-oriented data model as well as for other possible

data models [12]. He introduces the concept of only one additional normal form for objects as a substitute for fourth and fifth relational normal forms, having the following definition:

A collection of objects is in OONF if it is in 3NF and contains meaningful data elements only.

3.2 Khodorkovsky's ONF, 4ONF, 5ONF and 6ONF

The paper [8] proposes object normal forms, which concerns “the right relation” among objects and methods. The rules of defined object normal forms are based on modification of relational definitions of 4NF, 5NF (and 6NF, which is author’s original refinement of 5NF). The author calls these modifications of classical definitions as 4ONF, 5ONF and 6ONF.

The paper is considered to be more elaborated formulation of almost similar ideas as the example above. The author says, that 1NF, 2NF and 3NF are common for relational and object databases.

3.3 “Australian-Swiss” ONF

The authors [13] present only one ONF on various types of functional dependencies among objects. Concretely, “path dependency” concerns a composition of objects and navigability among objects, “local dependency” concerns relations of internal object and “global dependency” concerns behavioral requirements on application. Object-oriented structure is in ONF, if user requirements on applications are covered by a set of functional dependencies. This method relates to the behavioral requirement of object databases, but it is not specifically focused on the conceptual modeling of data.

3.4 Three Ambler-Beck's object normal forms

Ambler and Beck are pioneers of agile approach in programming. They introduced three object-oriented normal forms for object-oriented applications. [1,2,4]. These normal forms are analogous with first, second and third relational normal forms. The authors talk about these object normal forms as a tool for class structure normalization complementary with technique of design patterns. Let's look at their proposals in detail. Figures 1, 2, 3, and 4 are taken from [2].

A class is in 1ONF when specific behavior required by an attribute that is actually a collection of similar attributes is encapsulated within its own class. An object schema is in 1ONF when all of its classes are in 1ONF.

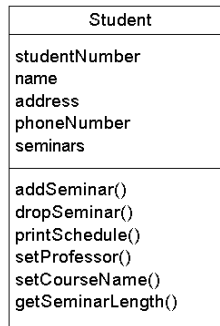


Fig. 1. 0ONF

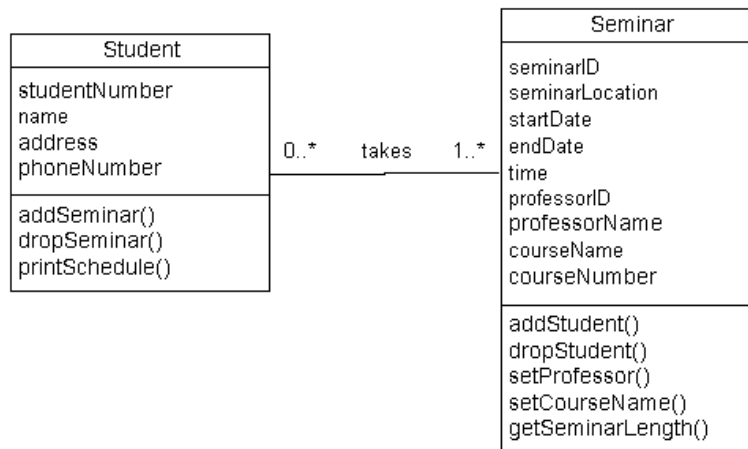


Fig. 2. Ambler's and Beck's 1ONF

It is evident from definition and example that authors wanted to build the first normal form analogically to the first relational normal form.

From experience, it is little confusing that object can stay non-normalized even if it already has associated collection of encapsulated objects. See attribute *seminars* of class *Student* in Figure 1. In this example the class *Student* contains the collection *seminars*, but the class *Student* is still in 0ONF. The collection *seminars* from 0ONF does not differ much from the relation *takes* in 1ONF in Figure 2. The difference between 0ONF and 1ONF is only in presence of specific methods of class *Student*.

A class is in second object normal form (2ONF) when it is in 1ONF and when "share" behavior that is needed by more than one instance of

the class is encapsulated within its own class(es). An object schema is in 2ONF when all of its classes are in 2ONF.

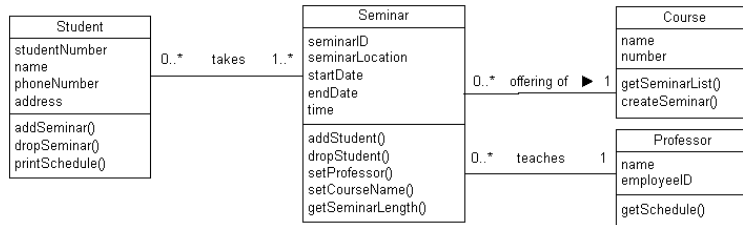


Fig. 3. Ambler's and Beck's 2ONF

As definition and example show in Figures 2 and 3, the 2ONF requires to detach attributes, which are shared by more objects, into separate objects. In our experience, this definition is well accepted. Also, this definition offers analogous result, as the second relational normal form.

A class is in third object normal form (3ONF) when it is in 2ONF and when it encapsulates only one set of cohesive behaviours. An object schema is in 3ONF when all of its classes are in 3ONF.

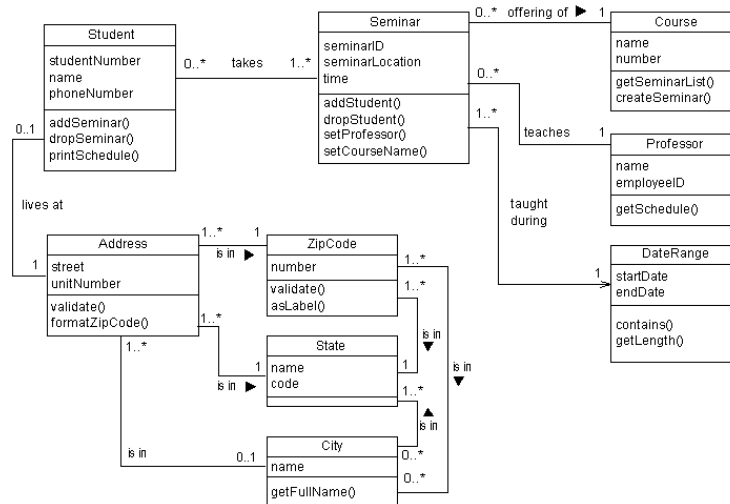


Fig. 4. Ambler's and Beck's 3ONF

In the third and the last object normal form by Ambler-Beck it is possible to recognize that it also give analogical results as the third relational normal form. This is our experience as well. It concerns the characteristics within some objects, which might be interpreted and behave as an independent object. In this case we need to exclude them into new separate object.

4 Our experience

We have good results with Ambler-Beck approach. But we have found that object-oriented community expects bit different technique:

1. It has to be very simple, and understandable and should work with minimum of abstract concepts, similarly as “the classical relational normalization”. We suppose that introduction of difficult definitions distinctively exceeding over the range of classical normal forms by having a lot of types of concepts and relations, is not the right way.
2. It should be focused concretely on object-oriented modeling of data structures. We need to model structures of objects used for data storage and data manipulation. We do not need to model objects responsible for functional behavior of applications. For these another “behavioral” objects we already have design patterns and other programming techniques. We do not need to duplicate these proven techniques. We think that original Ambler-Beck’s approach needlessly tries to solve everything in one.

We have to define, what we exactly understand by the concept of data object. Data objects serve only for data storing and manipulation. We will not work with data elements and with methods separately. This is proposed in [9]. We will define only one common concept of “an attribute”. By an attribute, we will understand the data property of an object regardless, if this data property is coming from a data element or if this data property is a result of a method.

Of course, there is a question, if such simplification is not too much. Ambler-Beck’s original approach works separately with data and methods and uses both of them separately. We believe that we can allow this simplification for the data objects, because our approach is not aimed for behavioral design of application structure.

4.1 First normal form rule

Definition 1. *A class is in the first object normal form (1ONF) when its objects do not contain group of repetitive attributes. Repetitive attributes must be extracted into objects of a new class. The group of repetitive attributes is then replaced by the link at the collection of the new objects. An object schema is in 1ONF when all of its classes are in 1ONF.*

More formally; Let us have an object a in the object system Ω as $a \in \Omega$, where for $k > 1$ (length of collections of similar attributes) and $n > 1$ (number of

repetition of these collections) is $data(a) = [\dots, x_1^1, \dots, x_k^1, \dots, x_1^n, \dots, x_k^n, \dots]$ having $\forall i \in (1, \dots, k) : class(x_i^1) = class(x_i^2) = \dots = class(x_i^n)$.

Then it is required to modify object a and create new objects $b_j \in \Omega$ for $j \in (1, \dots, n)$ as $data(a) = [\dots, \{b_j\}, \dots]$ and $data(b_j) = [x_1^j, \dots, x_k^j]$.

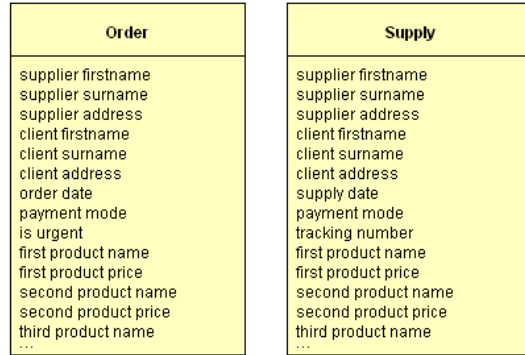


Fig. 5. Model before normalization

In the Figure 5 there is the example of data structure before normalization and in the Figure 6 there is the same example in 1ONF.

On the contrary with the original Ambler-Beck's approach, we do not assume analysts to recognize groups of repetitive attributes automatically and extract them out into independent classes. The problem is not always trivial as in presented example. Repetitive attributes can exist under various names, which are not easy visible on the first sight.

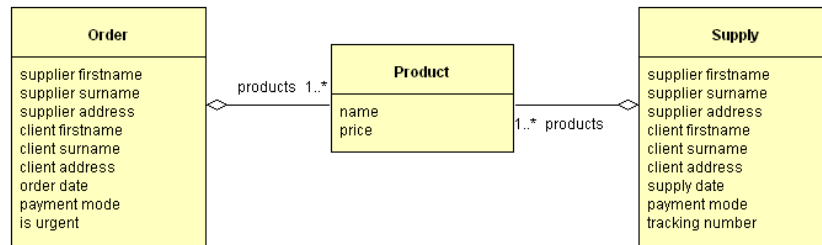


Fig. 6. Model in 1ONF

4.2 Second normal form rule

Definition 2. A class is in the second object normal form (2ONF) when it is in 1ONF and when its objects do not contain attribute or group of attributes, which are shared with another object. Shared attributes must be extracted into new objects of a new class, and in all objects, where they appeared, must be replaced by the link to the object of the new class. An object schema is in 2ONF when all of its classes are in 2ONF.

More formally; Let us have two objects $a, b \in \Omega$ for $k > 1$ (length of a collection of shared attributes) as $data(a) = [\dots, x_1, \dots, x_k, \dots]$ and $data(b) = [\dots, y_1, \dots, y_k, \dots]$ having $\forall i \in (1, \dots, k) : x_i = y_i$.

Then it is required to modify objects a and b and create new object $c \in \Omega$ as $data(a) = [\dots, c, \dots]$ and $data(b) = [\dots, c, \dots]$ and $data(c) = [x_1, \dots, x_k] = [y_1, \dots, y_k]$.

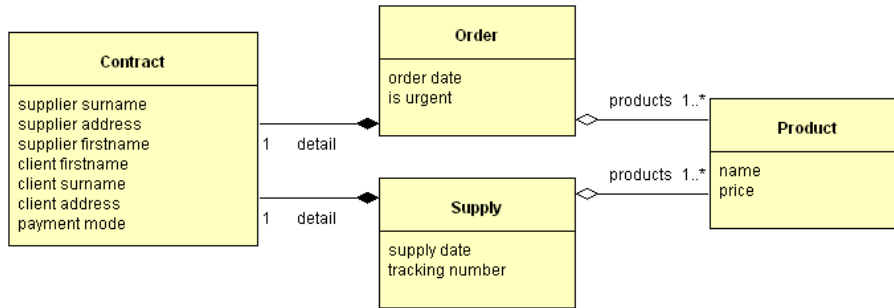


Fig. 7. Model in 2ONF

It concerns the attributes *supplier's first name*, *supplier's surname* and his *address* and *client's first name*, *client's surname*, his *address* and method of *payment* in our example. Because these attributes are common for both concrete order and supply, it was necessary to create the new object class *Contract*.

4.3 Third normal form rule

Definition 3. A class is in the third object normal form (3ONF) when it is in 2ONF and when its objects do not contain attribute or group of attributes, which have the independent interpretation in the modeled system. These attributes must be extracted into objects of a new class and in objects, where they appeared, must be replaced by the link to this new object. An object schema is in 3ONF when all of its classes are in 3ONF.

More formally; Let us have an object $a \in \Omega$ for $k > 1$ (length of a collection of independent attributes) having $data(a) = [\dots, x_1, \dots, x_k, \dots]$, where $[x_1, \dots, x_k]$ is collection of independent attributes.

Then it is required to create new object $b \in \Omega$ and modify object a as $data(a) = [\dots, b, \dots]$ and $data(b) = [x_1, \dots, x_k]$.

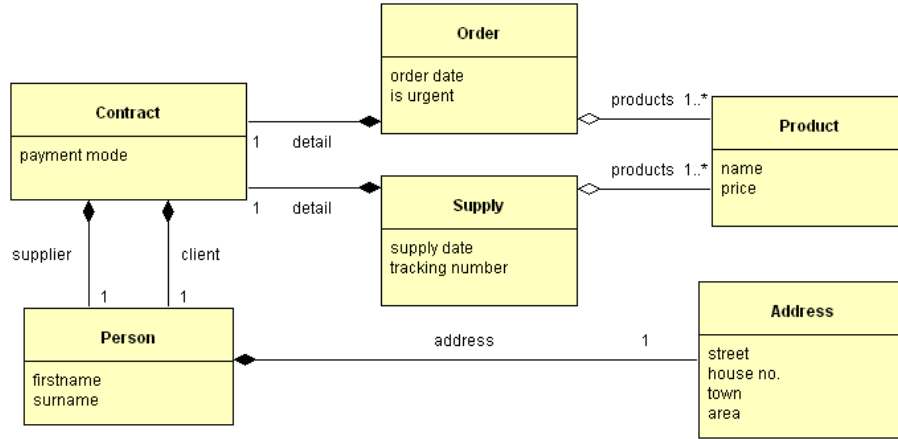


Fig. 8. Model in 3ONF

It concerns to the data about *suppliers* and *clients* in the objects of the class *Contract*. These attributes represent some *persons* having independent interpretation on contracts. The same applies to *addresses*.

4.4 Fourth normal form rule

Definition 4. A class is in the fourth object normal form (4ONF) when it is in 3ONF and when there is no other class in the system, which defines the same attributes. These attributes must be extracted from classes, where they are duplicated, and affected classes must be connected using class inheritance in order to exclude data definition duplicates. If there is no existing class to be reused as a inheritance superclass, a new superclass must be added into the system. An object schema is in 4ONF when all of its classes are in 4ONF.

More formally; For each two objects a, b in the object system Ω as $a, b \in \Omega$ having $data(a) = [x_1, \dots, x_k]$ and $data(b) = [\dots, y_1, \dots, y_k, \dots]$ where $\forall i \in (1, \dots, k) : class(x_i) = class(y_i)$, classes of these objects a, b must have inheritance relationship as $class(a) \prec class(b)$ in order to avoid duplicates.

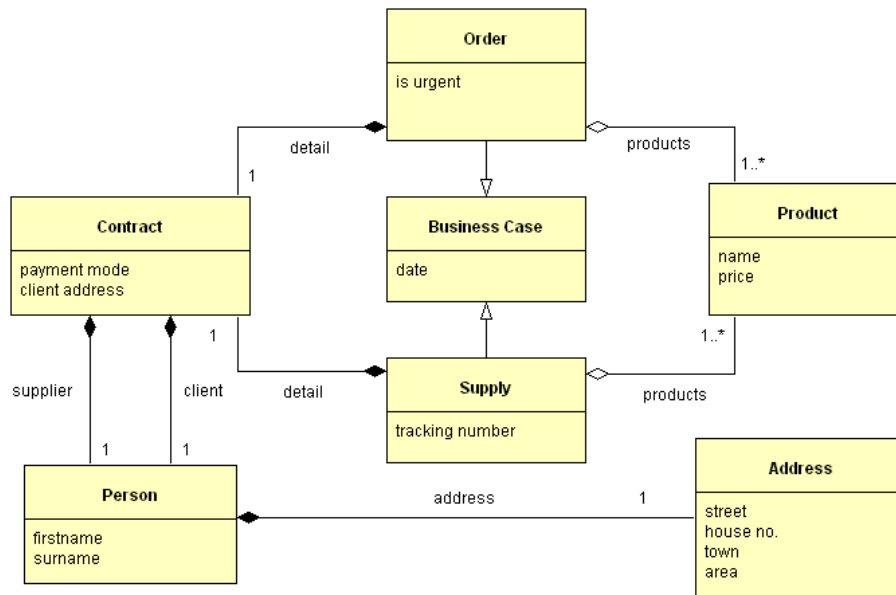


Fig. 9. Model in 4ONF

5 Conclusion

It is a pity, that so perspective and practically used technology – the object-oriented approach – still does not have comprehensible and universally accepted theoretical foundation and formal techniques. It is known, that several research centers are interested in this theme, but any coherent and widely accepted results were not yet published in recent years. Absence of reputable formal tools and techniques is the big problem of this promising technology. Therefore we suppose that near future may bring maybe some alternative approaches, more or less similar to our approach we presented in this paper.

Our method has been used in education at University of Thessaly in Volos, Alexandrian Technological Institute in Thessaloniki, Lehigh University in Pennsylvania, Czech Technical University and Czech University of Life Sciences. It was also used for database design in software development projects, which we carried out for a large international consulting company Deloitte. Recently, the project on object-oriented CASE tool supporting this approach sponsored by a consortium of software companies has been started.

Our future research will focus on describing the rules of our object-oriented normal forms as a sequence of refactoring steps.

References

1. Ambler Scott: *Building Object Applications That Work, Your Step-By-Step Handbook for Developing Robust Systems Using Object Technology*, Cambridge University Press/SIGS Books, 1997, ISBN 0521-64826-2
2. Ambler Scott: *Object Orientation - Bringing data professionals and application developers together*, <http://www.agiledata.org/essays/>
3. Barry D.: *The Object Database Handbook: How to Select, Implement, and Use Object-Oriented Databases*, ISBN 0471147184
4. Beck K.: *Agile Database Techniques- Effective Strategies for the Agile Software Developer*, John Wiley & Sons; ISBN 0471202835
5. Catell R. G.: *The Object Database Standard: ODMG 3.0*, ISBN 1558606475
6. *Gemstone Object Server* - documentation & non-commercial version download, <http://www.gemstone.com>
7. Goldberg, A. and Rubin, K. S.: *Succeeding with Objects - Decision Frameworks for Project Management*, Addison Wesley, ISBN 0-201-62878-3.
8. Khodorkovsky V. V.: *On Normalization of Relations in Databases*, Programming and Computer Software 28 (1), 41-52, January 2002, Nauka Interperiodica.
9. Kroha P.: *Objects and Databases*, McGraw Hill, London 1995, ISBN 0-07-707790-3.
10. Loomis M., Chaundri A.: *Object Databases in Practice*, ISBN 013899725X
11. MDA - The Model Driven Architecture, OMG - The Object Management Group (2009), <http://www.omg.org>
12. Nootenboom Henk Jan: Nuts - an online column about software design. <http://www.sum-it.nl/en200239.html>
13. Tari Zahir, Stokes John, Spaccapietra Stefano: *Object Normal Forms and Dependency Constraints for Object-Oriented Schemata*, ACM Transactions on Database Systems 513-569, Vol 22 Issue 4, December 1997.
14. *The UML standard*, OMG - The Object Management Group (2009), <http://www.omg.org>, ISO/IEC 19501.
15. Wai Y. Mok, Yiu-Kai Ng and David W. Embley, *An Improved Nested Normal Form for Use in Object-Oriented Software Systems*. Proceedings of the 2nd International Computer Science Conference: Data and Knowledge Engineering, Theory and Applications, pp. 446-452, Hong Kong, December 1992.
16. Yonghui Wu, Zhou Aoying: *Research on Normalization Design for Complex Object Schemes*, Info-Tech and Info-Net, vol 5. 101-106, Proceedings of ICII 2001, Beijing.