

Symmetries of Modelling Concepts and Relationships in UML - Advances and Opportunities

Vojtěch Merunka^{1,2}✉

¹ Department of Information Engineering, Faculty of Economics and Management,
Czech University of Life Sciences Prague, Prague, Czech Republic
vmerunka@gmail.com

² Department of Software Engineering, Faculty of Nuclear Sciences and Physical Engineering,
Czech Technical University in Prague, Prague, Czech Republic

Abstract. The article deals with the problem of asymmetry and redundancy of UML concepts that lead to overly complex modelling process without the possibility of efficient generalisation and decomposition of the modelled system to keep UML documentation to a manageable size, to improve user understanding, and simplify documentation and maintenance. The article includes a description of the original way of solving this problem.

Keywords: UML concepts and relationships · Symmetries · Diagramming techniques in UML · Decomposition in UML

1 Introduction - What Is UML?

Software engineers and practitioners are confronted with the question of what is actually UML. Is it mainly a standard of 14 UML-style diagrams or it is mainly a standard for modelling concepts, their properties and their mutual links, where diagrams are secondary as the most commonly used combinations of those more or less diagram-independent elements and links?

Of course, from a practical perspective, the UML standard is a standard for diagrams. But the idea of UML is larger. UML has its metamodel, UML also has its declarative programming language OCL, etc. UML has extension mechanisms (such stereotypes). Theoretically, we can imagine that we can have more new diagrams that would also respect the principles of UML. Moreover, maybe UML will absorb technology BPMN and yet some another tool.

But UML suffers by growing serious problems which were pointed by Simons and Graham (1999) several years ago. Leaving aside criticism of UML semantics, such as the direction of arrows, ugly aesthetics of some shapes, and other possibly confusing features of UML, we still need to stress following serious problems:

1. UML has too many species of terms in a very long but weakly-ordered list. Among these species, there is almost missing any taxonomy and hierarchy. It is in great contrast with the older techniques such Yourdon's structured method that sufficed

with easily logical and simple diagrams ERD and DFD having only a few of well-defined concepts. It is obvious that the UML is designed by engineers and practitioners, but not mathematicians. In short, the difference between UML diagrams and ERD & DFD is like the gap between the gigantic programming language C++ and smart programming language Scheme.

- UML has almost no support for the composition and decomposition of the modelled system. This problem is often multiplied at the enterprise level, where UML diagrams typically consist of hundreds or even thousands of elements. The only diagram that allows this property is a diagram of states and transitions. Instead of it, UML defines too complicated and confusing expansion-like relationships between various elements in different diagrams, such as in Fig. 1.

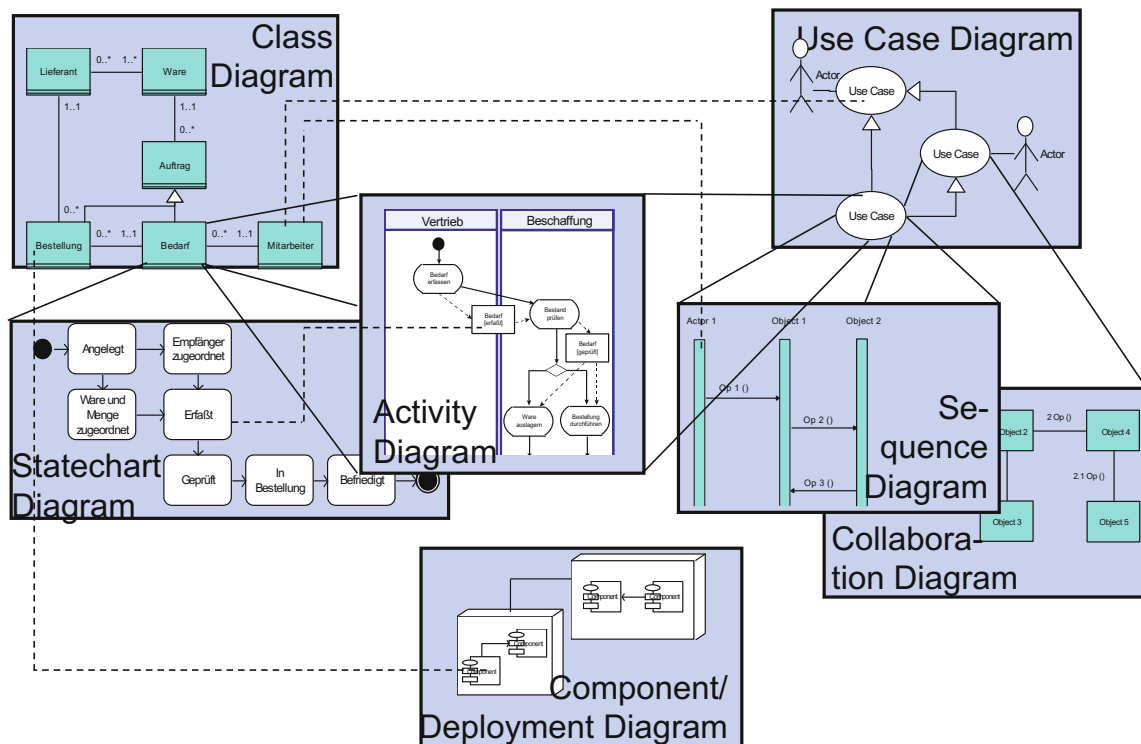


Fig. 1. Relations of UML diagrams and UML concept (Davis and Brabander 2007)

- UML has strong opposition in the agile programming community. These people say that they prefer «working software over comprehensive documentation» (AP manifesto 2001). The above problems of UML led these people to the radical conclusion that better than a bad conceptual documentation is no conceptual documentation. They decided to spend the saved time to full use of object-oriented development environments (e.g. refactoring, testing, code sharing) of platforms such as .NET, XCode, Ja-vaBeans, VisualWorks..., which are not sufficiently reflected in UML.

We dare to say that the limited possibility of making system composition-decomposition and the absence of symmetries among concepts in the UML is the greatest weakness of UML. Therefore, we think that UML will continue to swell more and more new concepts and will be yet more complex than now. UML is not able to keep pace with new

programming languages and new development environments in another way than by swallowing yet more new concepts, which degrade the original idea. UML is losing its potential power to motivate the evolution of development environments and languages in a similar way as UML predecessors Coad-Yourdon and Booch methods did.

2 The Motivation - Symmetries and Analogies in the Real World

In the physical world, the symmetry is an observable mathematical feature of a real system that remains unchanged under some transformation. The analogy is an effect of mapping some structure of one subsystem to another structure of another subsystem. Symmetries and analogies are maybe the essential principles how the God is building the world. Let's look at three examples of making systems more understandable using these principles (Table 1):

Table 1. symmetries and conservation law examples

Symmetry	Conserved quantity
Translation in time	Energy
Linear translation in space	Linear momentum
Rotation in space	Angular momentum

First example; Noether's theorem states that every symmetry of the action in a physical time-space system has a corresponding conservation law. Mathematician Emmy Noether proved this theorem in 1915 (Banados and Reyes 2016).

Second example; The Russian chemist Dmitri Mendeleev published the first widely recognised periodic table of chemical elements in 1869. He developed his table to illustrate periodic trends in the properties of the then-known elements, and he also predicted some properties of then-unknown elements that would be expected to fill gaps in this table. Most of his predictions were proved correct when the new chemical elements were subsequently discovered.

Finally; The Occam's razor is a general principle from philosophy which says if there exist more different models explaining some occurrence, the simplest one is usually the best one.

3 Solution

Fortunately, we have the hoped solution at our fingertips. Let us look at the UML standard primarily as a standard for software elements and their relationships, where UML diagrams are secondary. From this perspective, the UML sequence diagram and UML interaction diagram can be interpreted as the 2D projections of the single hypothetical 3D diagram, that shows everything in one (see Fig. 2). This is well known in better CASE tools (e.g. Visual Paradigm, Rational Rose,...) that allow creating an interaction diagram automatically from the current sequence diagram and vice-versa.

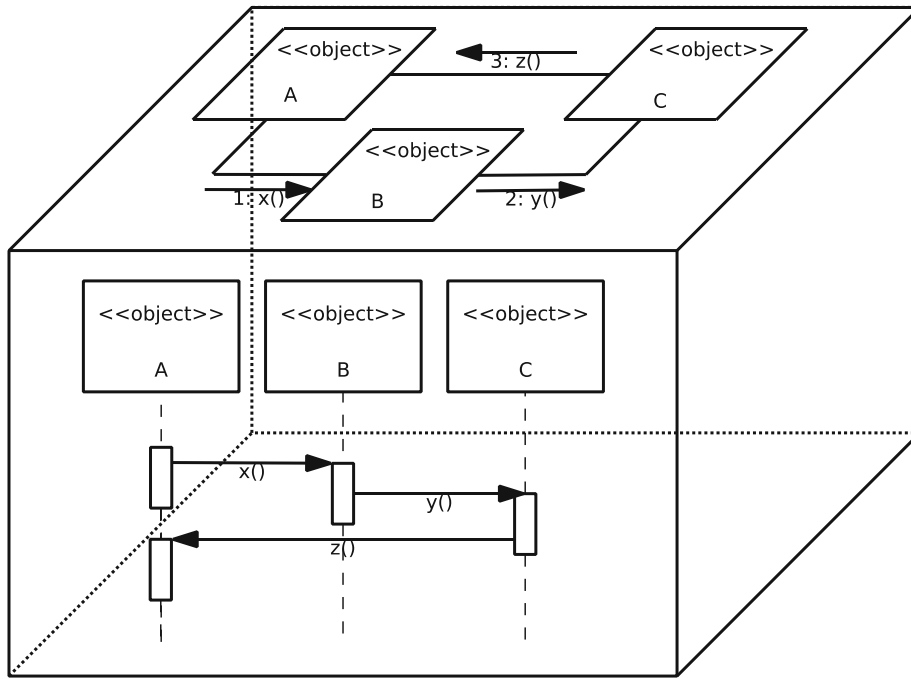


Fig. 2. 3D combination of a sequence and interaction diagram. [author]

If we generalise this effect to another UML concepts, we can combine also other UML diagrams in a similar way. In Fig. 3, there are two standard diagrams of states and transitions. The left diagram describes the behaviour of a borrower and right diagram describes the behaviour of a book in some library.

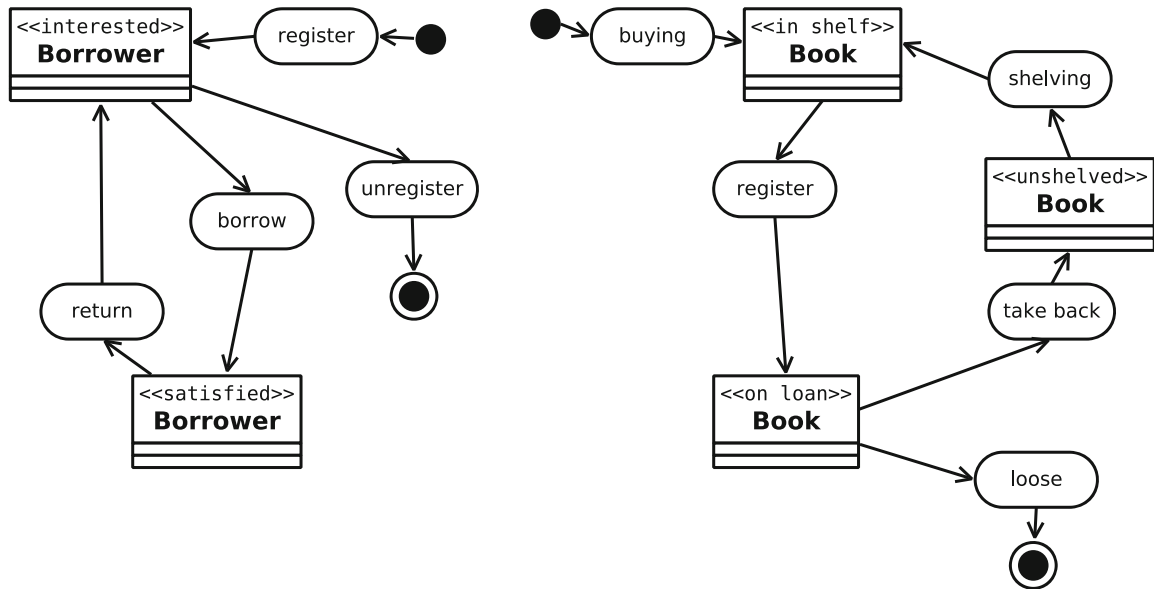


Fig. 3. State-transition diagrams of a borrower and a book in a library. [author]

If the object states will be considered as the greater detail of standard objects, then we can link them via associations, which we know from a standard UML class or instance diagrams. Likewise, based on the expected symmetry, we can also consider the

transitions between states as the activities (or methods), then we can connect these transitions via messages. It is interesting that as associations can have expressed their quantity as *cardinalities of associations*, also behaviours (e.g. messages) can have a similar concept, which we propose to call *frequencies of communications*. The overall result is shown in Fig. 4.

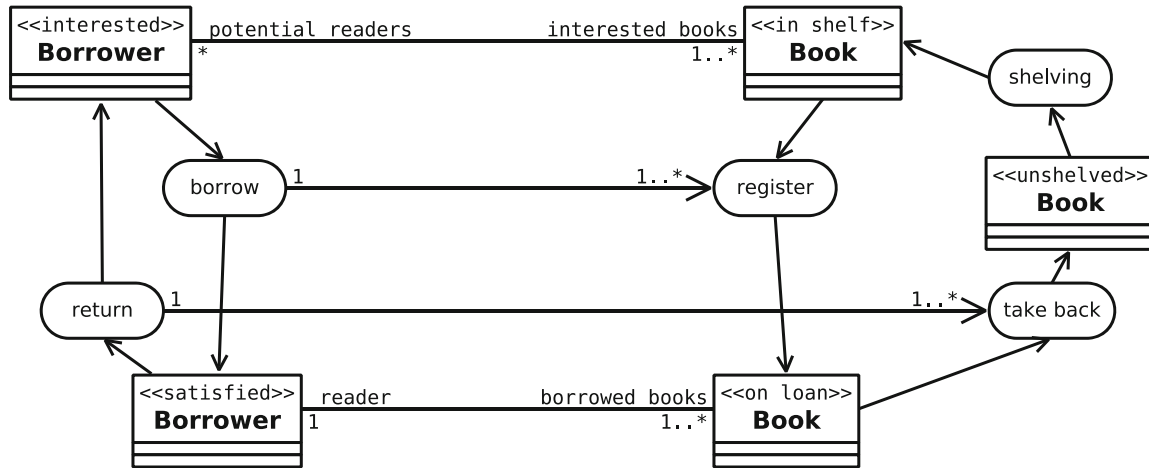


Fig. 4. Associations between states and communications between transitions. [author]

The composite model from Fig. 4 offers a very interesting option: Different states describe the behaviour of the same object during its whole existence, so we can create a simplified model that describes the same situation, but in the aggregate way of all individual time steps together. This *«all-in-one»* model is shown in Fig. 5.

This example shows that UML model can have *decompositions* via this generalization-refinement relationship, which is realised using the decomposition of the objects themselves into their states. Therefore, we can say that so much-needed general conceptual hierarchy of *generalization-refinement* is seamlessly present in the UML as well. It does not need to be the only *decomposition into states* of objects. Similarly, it may occur *decomposition into subtypes* (e.g. *is-a* hierarchy) or *decomposition into components* (e.g. *has-a* hierarchy) of objects.

When increasing the level of detail (e.g. refinement), we do not necessarily need to disaggregate all elements in the same level of detail. For example, if some of the objects are already implemented (e.g. reusable or legacy components), we can them filter out. Figure 6 brings such example, where, for some reason, the procedures *take back*, and *shelving* are not needed to be modelled in detail.

Decomposition of objects to states and transitions presented here is not the only possible way to introduce the decomposition into the object-oriented modelling. Objects can be decomposed (and aggregated back), also into the structures of multiple objects of more different classes, which are interconnected by the inheritance and whole-part relationships.

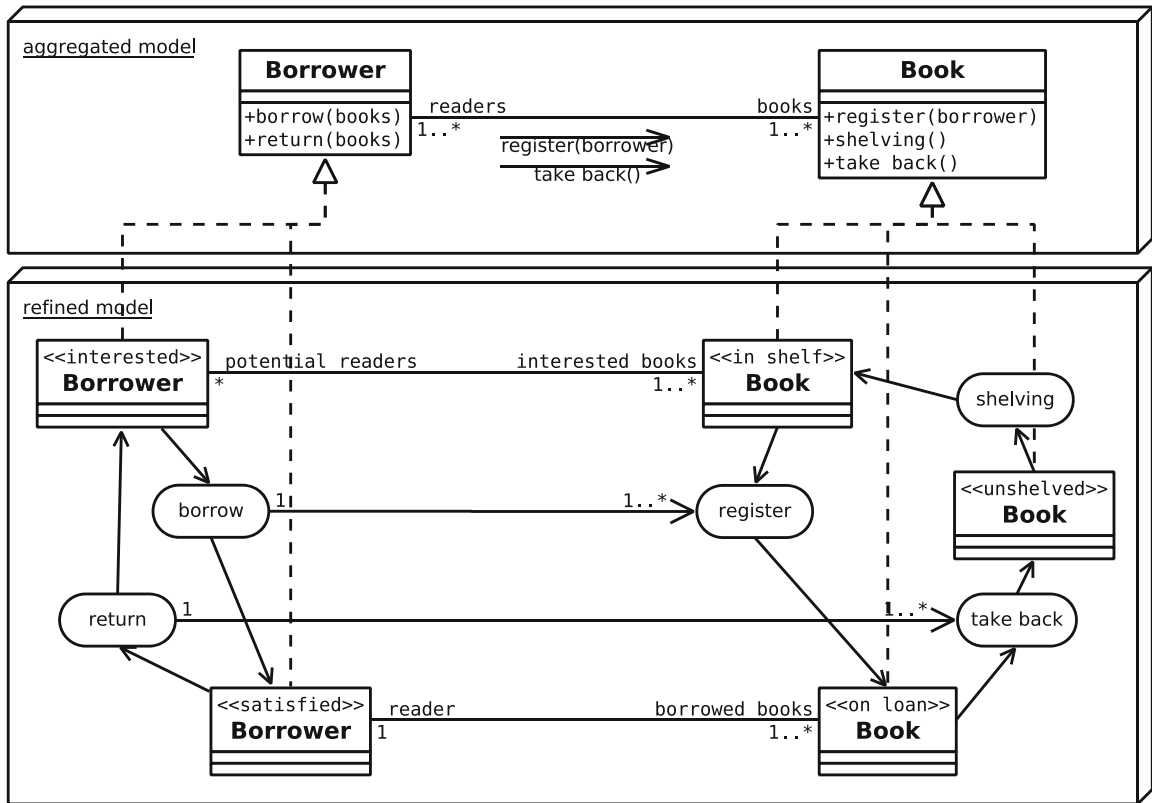


Fig. 5. Aggregation and refinement of the model. [author]

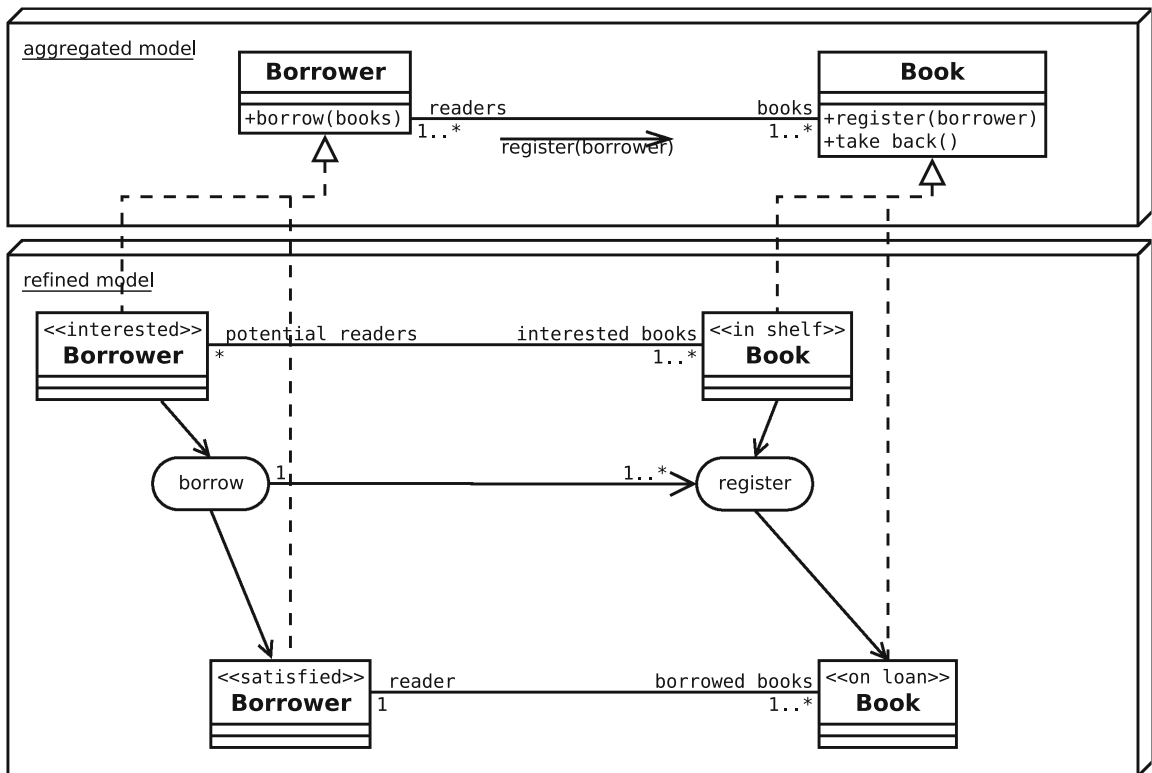


Fig. 6. Refined model with filtration of some unnecessary details. [author]

Another good candidate for this kind of UML model refinement is applying *design patterns* technique. *Design patterns* is already a technique which transforms CIM (*computer-independent model*) to PIM (*platform-independent model*) according to the principles of the MDA (*model-driven-architecture*) approach. The possibility of using *design patterns* technique in the process of MDA with UML is well described in the paper of Kardos and Drozdova (2010). Figure 7 presents an example of such transformation. Indeed, design patterns can now be understood as the tool of the model refinement to a higher level of detail. For example, a generalised model will have only one object class as the only entity, but its corresponding refined model will have this entity wrapped by some design pattern to ensure the requested system behaviour.

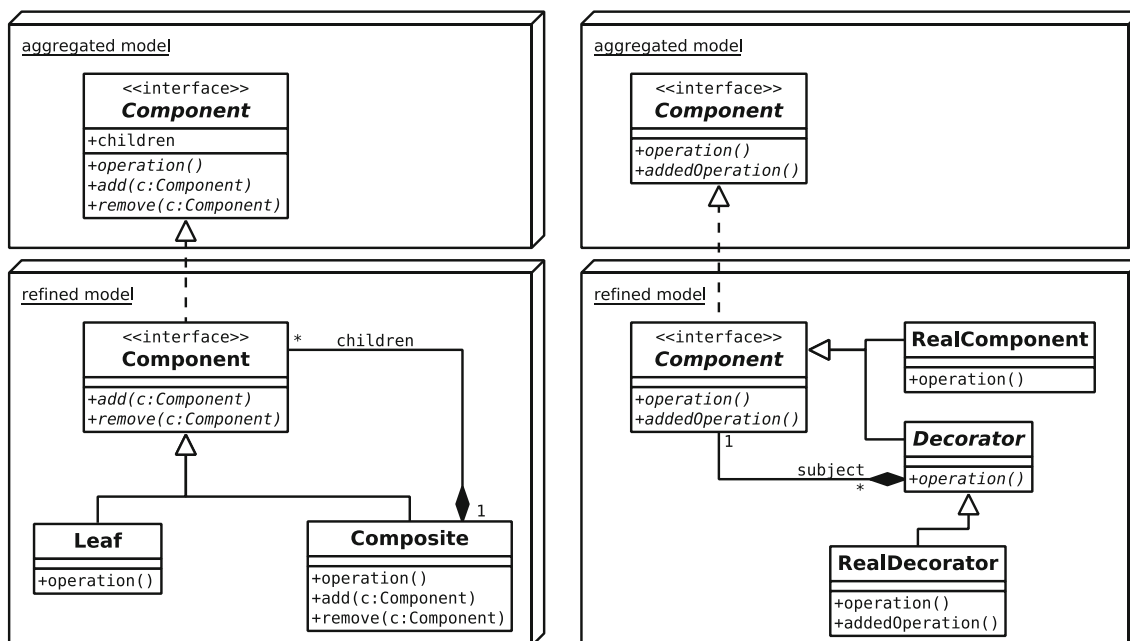


Fig. 7. Composite and Decorator design patterns as examples of model refinement. [author]

We should not forget yet one more thing. In 2005, UML was published by the International Organization for Standardization (ISO) as an approved standard ISO/IEC 19501:2005. It has become a universal tool for software engineering not only for the object-oriented programming but also for other programming paradigms. It means that UML also covers and even extends the full semantics of the old ER database diagrams. UML profiles for database modelling, including relational database technology, can be found for example in publications of Lo and Hung (2014) and Ambler (2003).

From this perspective, the semantics of UML class and object diagrams can be regarded as a superset (or evolution) of the former ERD. Therefore, it is very interesting that the authors Moody and Flitman (2000) did not try to apply their ERD decomposition principles to the UML class and object diagrams. We do not know any impediment, why their algorithm of data clustering cannot be applied to UML class and object diagram.

4 Discussion

Figure 8 gives a general overview of the proposed hierarchical approach to the UML modelling.

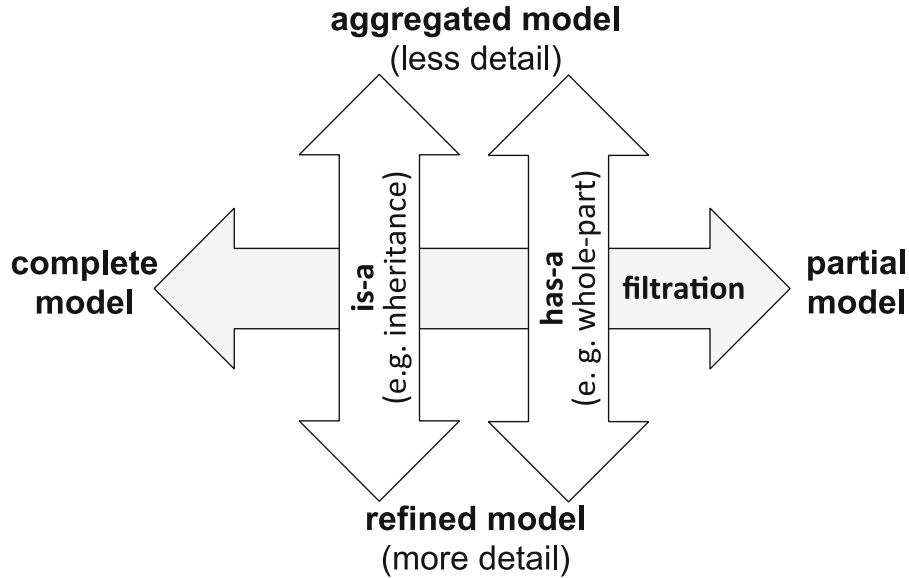


Fig. 8. Hierarchical approach to UML modeling. [author]

We have tried to describe a new approach in the form of a 2D table (see Table 2). This table has two major and mutually symmetrical rows, which correspond to the basic classification of UML 2.5 diagrams from Fig. 9. Four columns of this table show the scope from the highest level of abstraction (on the left) to the level of greatest detail (on the right). Of course, it would be possible to find yet more columns, but we think that

Table 2. Hierarchical approach to UML classification. [author]

Scope		System	Component	Object	Object interior	
Structure	Elements	system and actors	component	object	state	
	Relationships btw elements	actor - usecase link only	association	association	association with cardinalities	
	Hierarchies	is-a	supertype-subtype of actors		inheritance of objects	
		has-a	actor or subsystem whole-part	package decomposition	whole-part links of objects	state diagram decomposition
Behaviour	Elements	use case	interface	method	transition	
	Relationships btw elements	actor - usecase link only	connectors and ports	message	<i>communication with frequencies</i>	
	Hierarchies	is-a	usecase link «extends»	interface inheritance	inheritance of methods	
		has-a	usecase link «includes»		method code decomposition	state diagram decomposition

our four columns (*system - component - object - object interior*) will be entirely sufficient to illustrate our approach.

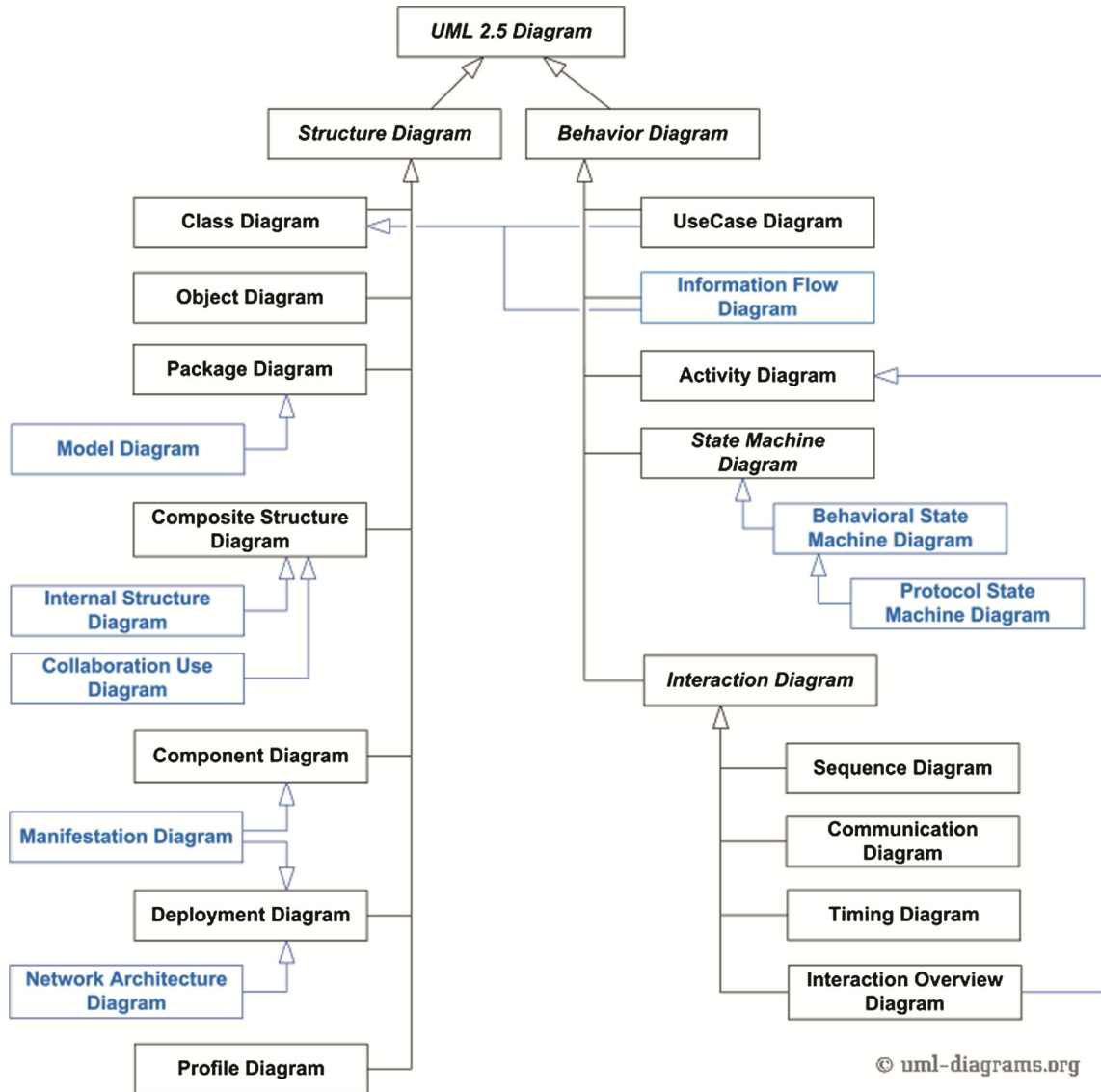


Fig. 9. UML 2.5 diagrams. [<http://www.uml-diagrams.org>]

4.1 New Concept of Communication with Frequencies

The newly designed concept of *communication* with *frequencies* between transitions (implemented by object methods) is in the symmetrical position with the concept of *association* with *cardinalities*. It should be noted that similar frequency (or cardinality) can be found in the links between actors and usecases in the detailed use-case diagrams. For us, it is not surprising, because the actor - use-case links are regarded as the highest-level generalisation of the communications with frequencies. Even, frequencies can also be added to messages and links between connectors and ports, where should also make sense. Analogously, we can anticipate the inclusion of other new concepts. The empty cells in Table 2 are candidates for these new concepts.

4.2 Solution of the Problem with Association and Whole-Part Relationship

In UML, the concept of a *whole-part* relationship seems to be a special kind of *an association*. Novices in the UML modelling have problems because they do not know which link you choose in a particular case. Similarly, this mistake often happens to professionals having years of experience in programming; database programmers tend to prefer *associations* everywhere, but object-oriented programmers tend to prefer *whole-part object relationships*. The most used database technology is based on the concept of the relation between data, but the object-oriented programming directly supports only object whole-part relationships. Object-oriented programming technology and the most used database technology are real, everybody is very likely working with both, and they are here to stay and must be supported by the UML. Unfortunately, these two programming technologies have a difference which is called «the object-relational impedance mismatch» (Ambler 2013).

Our solution is following:

- Objects in the *whole-part* relationship are in a dependency relationship like the object *inheritance* relationship. Both *whole-part* and *inheritance* relationships (better naming would be *hierarchies*) are used to describe some higher or lower level of abstraction and are subject to aggregation/refinement procedures.
- *Associations* between objects are the relationship between the independent elements of the same level of abstraction. Associations are subject to aggregation/refinement procedures, only if their objects have own hierarchies.

5 Conclusion

This paper has presented an alternative approach to classification of UML concepts and their relationships. This approach enables a manual procedure for decomposing and composing UML models into hierarchies of manageable size, which allows a human to improve the quality of the result, reduce uncertainty and improve conceptual consistency among members of the development team.

The major theoretical contribution of this paper is an alternative perspective on the current UML standard, which provides a solid foundation for both future theoretical research and practical implementation of new CASE tools.

1. We confirmed the roles of the *inheritance* and the *whole-part* relationship as tools for *generalisation* and *refinement* of UML models.
2. We explained the conceptual difference between *association* (which is not directly included in object-oriented programming languages) and *whole-part* relationship (which is directly included in object-oriented programming languages).
3. Based on symmetry with *associations* and *cardinalities* which inform about the quantity of *association* relationship between object species, we recognised the existence of *communications* and *frequencies* which inform about the amount of *communications* between object behaviours.
4. We demonstrated that UML does not need to grow and mindlessly absorb new concepts at any cost but just refine what is already done.

Our future research will focus on the empirical justification of our statements and find new algorithms of model transformation that would automatically perform proposed model transformations.

Acknowledgments. This paper was elaborated under the support of the grant project SGS17/197/OHK4/3T/14 of the CTU in Prague.

References

- Agile programming manifesto (2001). <http://agilemanifesto.org/>
- Ambler, S.W.: Agile Database Techniques: Effective Strategies for the Agile Software Developer. Wiley Publishing Inc., New York (2003). ISBN 978-0-471-20283-7
- Ambler, S.W.: The Object-Relational Impedance Mismatch, in Agile Essays by AmbySoft Inc. (2013). <http://www.agiledata.org/essays/impedanceMismatch.html>
- Banados, M., Reyes, I.: A short review on Noether's theorems, gauge symmetries and boundary terms. Universidad Catolica de Chile (2016)
- Davis, R., Brabander, E.: Aris Design Platform, Chapter 2 - UML Designer (2007). ISBN 978-1-84628-612-4
- Kardoš, M., Drozdová, M.: Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). *J. Inf. Organ. Sci.* **34**(1), 89–99 (2010)
- Lo, C.-M., Hung, H.-Y.: Towards a UML profile to relational database modelling. *Appl. Math. Inf. Sci.* **8**(2), 733–743 (2014). <http://dx.doi.org/10.12785/amis/080233>
- Moody, D.L., Flitman, A.R.: A decomposition method for entity relationship models: a systems theoretic approach. In: Proceedings of the First International Conference on Systems Thinking in Management, Geelong, Australia (2000)
- Simons, A.J.H., Graham, I.: 30 Things that Go wrong in object modelling with UML 1.3. In: Kilov, H., Rumpe, B., Simmonds, I. (eds.) Behavioral Specifications of Businesses and Systems. The Springer International Series in Engineering and Computer Science, vol. 523, pp. 237–257. Springer, Boston (1999). doi:[10.1007/978-1-4615-5229-1](https://doi.org/10.1007/978-1-4615-5229-1)