Modelling Life Cycles of Generic Object Classes

Vaclav Repa

Abstract The paper analyses the problem of class generalisation from the non-traditional point of view—class life cycles. Model of class life cycle represents a process-oriented view on the class which is not usual in the field of conceptual modelling. This paper focuses on the problem of modelling life cycles of generic classes when there is a need to model several life cycles, valid at the same time. The paper shows that this problem is rooted in the natural contradiction of object- and process-oriented approach to modelling following from the fact that these two basic approaches are based on mutually contradicting primary types of hierarchical abstraction. The paper also shows that this problem is closely connected with the "problem of conflicting identities" in generalisation trees discussed on the border of the conceptual modelling and ontology engineering fields.

1 Introduction: Historical Background

The latest development of the *enterprise architecture* methodologies is strongly influenced by the need for modelling the real-world aspects of an enterprise as a universal base for the further design. The basic technique for modelling the real world, representing the basic approach to the analytical thinking in the field of informatics, is the *conceptual modelling*. The concept "conceptual" has been firstly used in the area of data modelling. It expresses the fact that the database should describe the essential characteristics of the real world: objects and their mutual relationships. This origin is still visible in common understanding of the adjective "conceptual" in the sense of modelling with the Unified Modelling Language [10]:

V. Repa (✉)
Department of Information Technology, University of Economics, Prague,
W. Churchill sqr. 4, Prague 3, 130 67, Czech Republic
e-mail: repa@vse.cz

H. Linger et al. (eds.), *Building Sustainable Information Systems: Proceedings of the 2012 International Conference on Information Systems Development*,
DOI 10.1007/978-1-4614-7540-8_34, © Springer Science+Business Media New York 2013

Craig Larman [8] describes conceptual modelling such as the following: classes represent concepts from the real-world domain, binary associations describe relationships between two concepts, and the concepts can have attributes but no operations. Cris Kobryn [6] speaks about "structural model" which shows the static structure of the system: the entities that exist, internal structure, and relationships to other entities. Roni Weisman [12] uses the term "conceptual system model" and distinguishes three types of objects: entity (objects which hold the system's data), boundary object (interface objects which directly interact with the external world—actors), and control object (objects which manage the system operations). Although there are various approaches to the conceptual modelling in object-oriented methods, each of them reduces the conceptual model (represented by the class diagram) to the model of objects and relationships between them, represented by their attributes but not by their methods. This reduction is present even in Weisman's approach (see above) where only "entities" represent the real-world objects while "control object" expresses rather the behaviour of the "system". The common understanding of the term "conceptual" thus tends to be the synonym for "static".

However, such an approach contrasts with the basic principle and the main contribution of the object-oriented paradigm—unity of data and operations. This principle evokes the idea that it is necessary to model not only static aspects of the real world but also its dynamics. Thus, not only attributes but also relevant object's operations together with their essential time consequences should be regarded as a property of the real world. Such description of the essential dynamics of the object is usually called the object's life cycle.

The interest in object life cycles originated in the historical context of business process modelling in the early 1990s. According to Kappel and Schrefl [5], *an object life cycle is a model that captures allowed states and state transitions for a particular object type*. A common means for modelling life cycles of objects is a non-deterministic finite state machine. The UML contains for this purpose the state chart (state machine) diagram. Other UML diagrams for description of dynamics (activity diagram, sequence diagram, objects interaction diagram) are also usable but not so suitable for this purpose because they are not so closely and explicitly connected with the class diagram as a crucial diagram for the conceptual modelling (see the further argumentation below).

The historical context of the business process modelling, mentioned above, also brought the general, still persisting, erroneous impression that business processes can be modelled via life cycles of participating objects. Küster et al. [7] clearly identified the substantial difference between an object life cycle and a business process formulating the set of rules for "generation of a compliant business process model from given object life cycles". Although this approach views the business process too mechanically, almost as a mechanical consequence of business rules expressed by object life cycles, the identified difference is a very significant shift in the perception of essence of object life cycles.

The motivation for modelling life cycles of classes presented in this paper comes from the methodology for information modelling of organisations [9]. This methodology is based on systematic work with two parallel model dimensions, conceptual

(object oriented) and behavioural (process oriented), and distinguishes between the two main types of processes which are to be described in order to fully define the nature of the real world:

- Business (intentional) processes
- Life cycle (non-intentional, substantial) processes

The *business process* always represents some intention, expresses the way of achieving some goal, and has some products, and it is typical expression of the human will.

On the other hand, the *life cycle of an object* has no goal, nor product; it is rather the expression of the objective necessity, usually called business rules. Objects are typically taking different roles in different processes giving them the context (real-world rules), while business process typically combines different objects giving them the specific meaning (roles of actors, products, etc.).

As it follows from the previous paragraph, the objects life cycles cannot be regarded as business processes but rather as a *description of business rules in a process manner*. The presented methodology uses for the modelling objects life cycles the state chart from the UML which is, according to the UML metamodel [11], principally connected with the class diagram via the concepts of class, method, and some others. This connection is also well visible in the Business Substance Metamodel from the Business System Metamodel [1] as a part of the OpenSoul Methodology where the UML metamodel is extended with the concepts of class state, class life cycle step, and class life cycle which address exactly this methodical consequence.

The example in Fig. 1 illustrates the state chart describing the life cycle of the object *order* as a complement to the class diagram which describes the context of

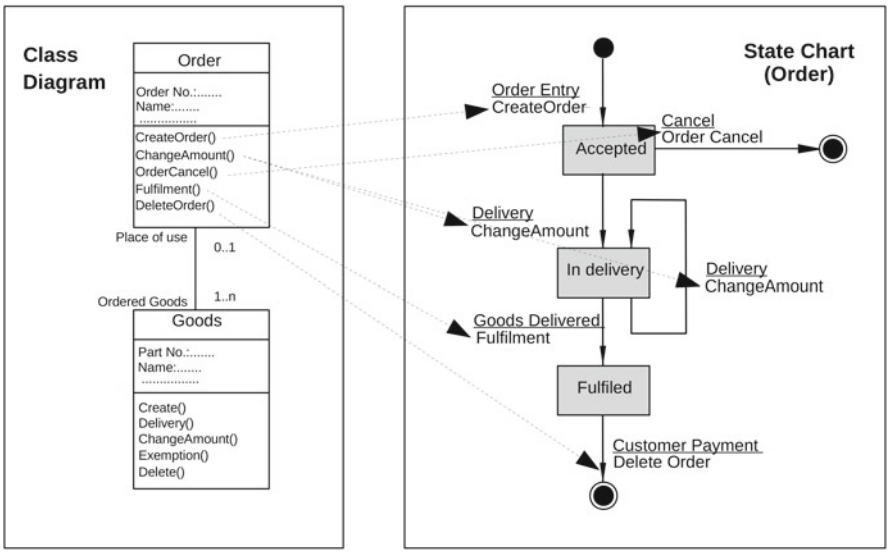

Fig. 1 Life cycle of the class

this object. All methods of the conceptual object should be ordered into one algorithm which defines the "conceptual meaning" of each method as its place in the process of the object's life.

Generalisation—a natural way of building the hierarchy of concepts—is one of the most significant concepts in the field of conceptual modelling. This way of organising concepts is also used as a basic tool for the top-down process of the analysis of concepts. On the higher level of abstraction, we work with global—generic concepts and their general associations, then on more detailed level we can analyse detailed—more specific types of previously analysed concepts and their specific relationships. Generalisation as an essential way of building hierarchies of concepts forms the roots of conceptual (alias "object oriented") modelling languages. In the UML the generalisation occurs as one of basic principles hidden in the so-called principle of inheritance.

Combining the above discussed need for modelling the dynamics of objects together with the importance of generalisation as a natural way of building the hierarchy of concepts, we can formulate the crucial question: *how to model life cycles of generic classes*?

In the following text, we will discuss the problem of generalisation of classes from a non-traditional (class life cycles) point of view. We will focus on the problem of modelling life cycles of generic classes when there is a need to model the life of the generic class together with the life of its specific subtype at the same time. This problem is rooted in the natural contradiction of object- and process-oriented approach to modelling following from the fact that these two basic approaches are based on mutually contradicting primary types of hierarchical abstraction. We also show that this problem is closely connected with the "problem of conflicting identities" in generalisation trees discussed in several works in the fields of conceptual modelling as well as ontology engineering.

In the first section, we will pay attention to the problem of using generalisation in conceptual models. We will formulate the problem of modelling life cycles of different concepts representing the same object in the generalisation tree, and we show that this problem has the common root with the language insufficiencies discussed before. In the last section, we will summarise previous sections and formulate basic conclusions.

2 Generalisation from the System Versus Process Point of View

Generalisation as a principal way of hierarchical classification of concepts is widely used in the conceptual modelling based on modal logics. Giancarlo Guizzardi [3] convincingly shows the necessity of the classification of different types of specialisation of concepts which occurs especially in the case of subtypes of the so-called <<phase>> type (see the example in Fig. 2). The problem is that the particular instance of the class *customer* can be over time of different subtypes *potential*,

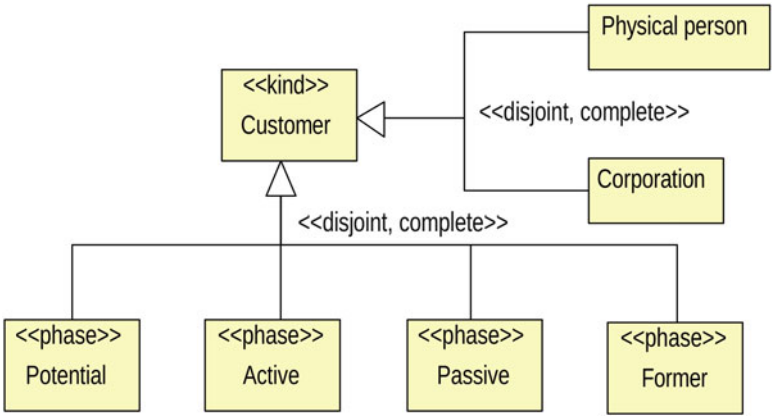

Fig. 2 Ontologically correct multiple specialisation of the class customer

active, *passive*, and *former* staying the same individual. Thus, these subtypes do not represent different instances of the class *customer* but rather different states (phases) of the same instance. Guizzardi also notes that the same problem is addressed by Heller and Herre in [4] who, in generalisations, distinguish between abstract substances called *persistents* and so-called presentials which represent momentary states.

In this example we pragmatically reduce our universe of discourse ignoring the general fact that the *customer* is rather the role of some general concept (person for instance) instead of the standalone kind as it is regarded here. In our real world, the *customer* is a substantial primary concept and we do not analyse its general superclassifications as it is not relevant there.

Figure 2 shows the nonconflicting multiple specialisation of the concept *customer*. From the point of view of the natural evolution of general *customer*, it is important to distinguish among its particular phases *potential*, *active*, *passive*, and *former* which the *customer* generally can pass over time. Nevertheless, it is also important to distinguish between two general types of *customer*: *physical person* and *corporation*, simultaneously. As the second classification (*corporation/physical person*) is not transitive, i.e. the particular instance of customer cannot transit from one type to another during its life, there is no danger of possible conflict of this classification with the first one (evolution phases of *customer*). Both classifications can exist simultaneously without mutual interference because they are independent in principle. The mechanism of evolution of a *customer* is exactly the same whether it is a *physical person* or *corporation*, and *customer* is either *physical person* or *corporation* no matter in which evolution stage it is.

If there is a need for time dependent—<<*phase*>> specialisation of a class—it can be supposed that there is also a need for detailed description of general conditions and circumstances for the transitions between its particular phases. Such

description can be done as a so-called life cycle of a class where above mentioned conditions and other general circumstances are described as a process of possible transitions among identified states of a class.

While modelling life cycles of both generic classes and their specific subtypes, we always need to handle the following problem: we should express the common structure expressing both generalisation and aggregation. This problem is also addressed by Ebert and Engels in [2] where they pointed to the need to reconcile the conceptual modelling with modelling of life cycles in the case of generic objects. It is rooted in the natural contradiction of object- and process-oriented approach to modelling. This contradiction follows from the fact that each of these two basic approaches have a different primary type of hierarchical abstraction. In the object-oriented approach, hierarchy means primarily generalisation; while in the process-oriented approach to modelling, hierarchy means primarily aggregation. As these two basic types of hierarchical abstraction are mutually exclusive (the hierarchy can express either generalisation or aggregation), object- and process-oriented approach to modelling are in principal contradiction. Practically, this means that it is impossible to express all crucial process aspects of a reality by means of objects only (alias to model processes as objects and their relationships) as well as it is impossible to express all crucial object aspects of a reality just by means of processes (alias to model objects as processes and their successions).

In the object-oriented approach, the generalisation exists as a principle (so-called inheritance principle), and the second possible type of hierarchical abstraction—the aggregation plays the role of just a specific type of relationships of objects. In the process-oriented approach, the aggregation exists as a principle (the rule that "any activity as a part of a process can be taken as a standalone process on a deeper level of detail"), and generalisation plays a secondary role of just a specific kind of relationships of different, mutually exclusive, activities. Above discussed facts mean that to express the life cycles of both generic and specific classes of the same generalisation tree we have to describe the generalisation tree as a set (i.e. aggregation tree) of processes.

Figure 3 describes the life cycle of the generic class *customer* from Fig. 2. According to the methodology, each phase—specific subtype of the generic class—is represented by the specific state of its life cycle. The life cycle thus describes the general process of the class metamorphosis from one subtype to the others.

The example in Fig. 3 specifies the general mechanism of possible changes of the states (alias subtypes) of the object *customer*. Each transition between states has its reason (real-world event) and is performed by the action which belongs to the class (class method).

If needed, it is possible to describe also specific structure of the life of the object in the specific state as it is illustrated in Fig. 4. This allows expressing the generalisation structure of concepts as an aggregation of life cycles. Nevertheless, the very important fact, and the fatal condition of this situation, is that the whole generalisation is of the <<*phase*>> type (i.e. all subtypes are of this type of specialisation).

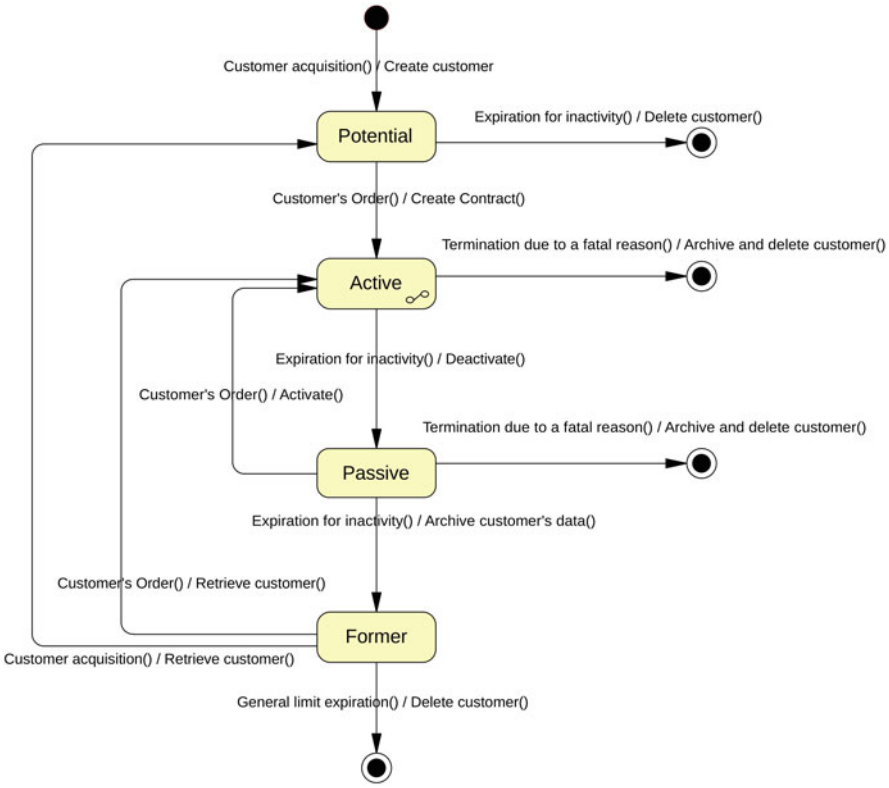

Fig. 3 Life cycle of the generic class customer

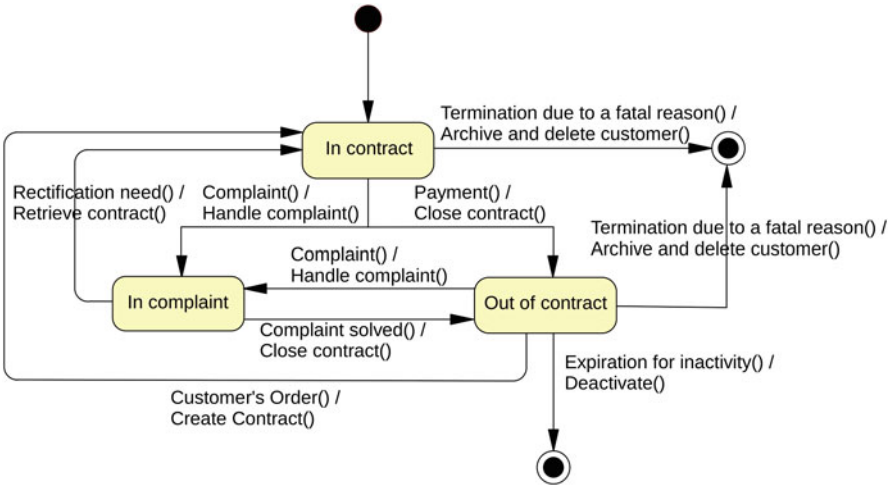

Fig. 4 Life cycle of the specific—active—type of the generic customer

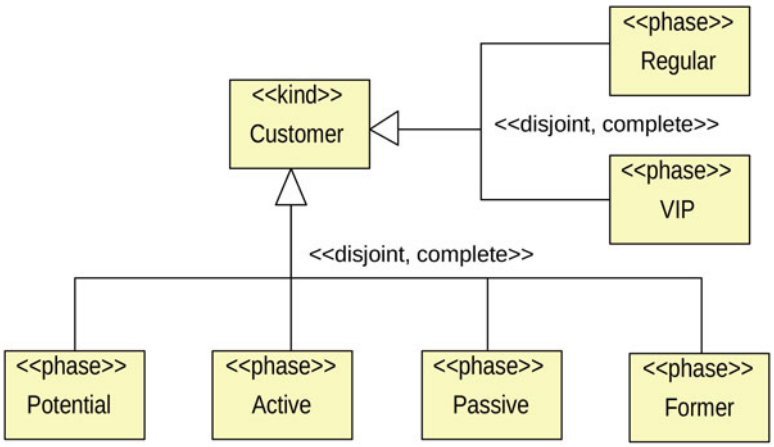

Fig. 5 Conflicting "phase" specialisations of the class customer

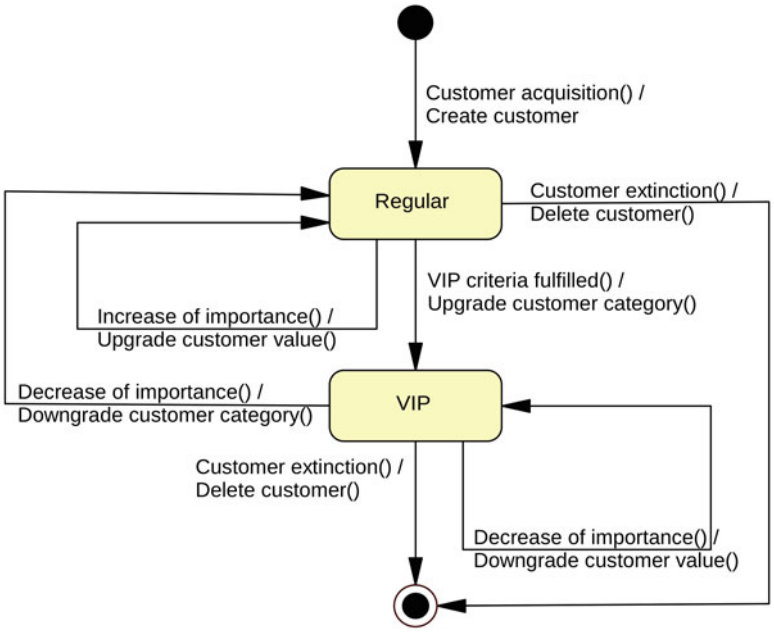

Fig. 6 Life cycle of the class customer from the importance point of view

The second crucial problem connected with describing life cycles of generic object classes occurs in the case of multiple <<*phase*>> generalisation trees for the same class. Such situation is illustrated by the example in Fig. 5.

In this situation there is the additional valid life cycle of the same class customer besides the life cycle described in Fig. 3. This additional life cycle is described in Fig. 6. The problem is that these two life cycles must be valid at the same time as it

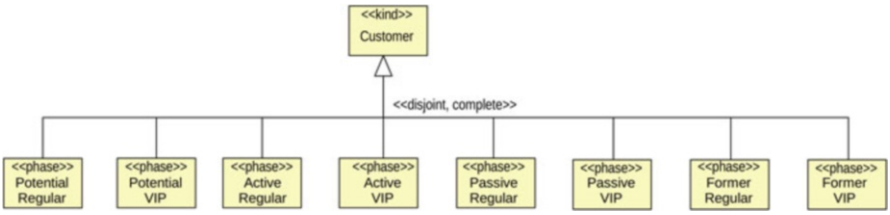

Fig. 7 Possible solution of the conflict of phases via the Cartesian product of possible phases

belongs to the real-life situation like the example illustrates. There are specific combinations of specific states of different types for the same object. At the particular moment, just one specific combination is valid.

The model in Fig. 7 expresses a virtually sufficient solution of the problem of conflicting "phase" specialisations of the class *customer* from Fig. 5. Both classification structures are combined in the Cartesian product and the result is one specific subtype for each combination of each two possibilities. From the global point of view (represented by the class diagram), it seems that the problem is no more existing; the only structure of the subtypes naturally prevents any possible conflict of identity, and the completeness is guaranteed by the combination of all possibilities.

The problem connected with the specialisation in Fig. 7 is that it is principally impossible to express exactly all combined phases as the states of one single life cycle (i.e. in one common state chart). Although from the system point of view (class diagram), each combination of states looks real, when we look at these combined states regarding the factor of time, we can realise that:

(a) Some combinations cannot exist in reality because they do not make sense (e.g. if there is a rule that the customer without an order cannot be *VIP*, the subtypes *Passive VIP* as well as *Potential VIP* are not real).
(b) Some combined states represent rather an aggregation of really elementary states of this type (from the time point of view) than a single state (i.e. in a single moment). This fact causes the possible situation where different states can exist at the same time. For example, the states *Former VIP* and *Former Regular* cannot be clearly distinguished as this single customer could be several times VIP and several times regular in history. If the states cannot be clearly distinguished it is impossible to express their ordering on the time line. Such a situation usually signals that there are more different objects which the states belong to; thus, it is not possible to describe these states as elements of the single algorithm (life cycle).

While the problem (a) is not critical because it can be easily solved by omitting non-relevant states, the problem (b) is fatal. It is a proof that the solution of the conflict of phases via the Cartesian product of the possible phase combinations is principally wrong.

The correct solution of the conflict of phases from Fig. 7 is described in Fig. 8. It is necessary to respect the fact that these two <<*phase*>> specialisations (*potential*, *active*, *passive*, *former* versus *regular*, *VIP*) are independent in principle and cannot

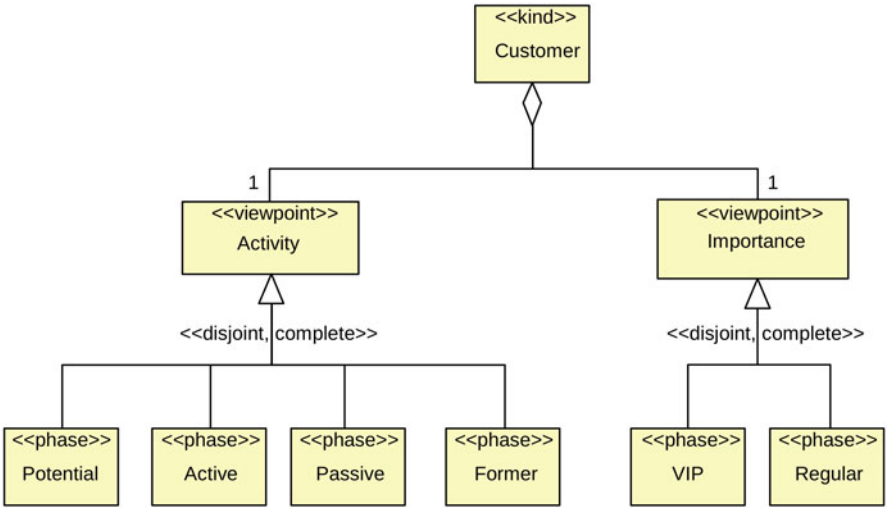

Fig. 8 Generally correct solution of the conflict of phases with <<*viewpoint*>> decomposition

thus be combined. Therefore, we introduce abstract concepts *activity* and *importance* which represent two different points of view which are, in fact, represented by these two independent classifications. Each new generic concept represents the part of the super-concept *customer* from its point of view.

For this general-methodical solution of the problem of multiple <<*phase*>> generalisation trees, we introduce a new type of abstract class <<*viewpoint*>> existing under following rules:

(a) <<*viewpoint*>> must be an abstract generic class representing specialisation tree of the <<*phase*>> type.
(b) <<*viewpoint*>> class never represents the real object.
(c) <<*viewpoint*>> must be a part of an aggregation structure. This aggregation represents an abstract class which is a generic concept representing all phases of all its components, and which:

- Cannot be of the <<phase>> type
- Always represents the real object

The purpose of this methodological construction is to allow expressing two crucial facts:

- The abstract class which is a head of the structure (the concept *customer* in the example) is specified with multiple life cycles which are mutually independent.
- All the specified life cycles of the class are valid at the same time.

This way the problem is definitely solved. Each of the both life cycles is guaranteed valid (independently of the second one), and both are valid at the same time as it is warranted by the aggregation of both viewpoints.

3 Summary and Conclusion

In this paper we have discussed the problem of modelling life cycles of generic classes. We have identified two basic types of problems which can occur:

- The problem of aggregated life cycles if there is a need to model the life of the generic class together with the life of its specific subtype at the same time
- The problem of multiple <<*phase*>> generalisation trees for the same class if there is a need to model the life of the generic class from different points of view which all are valid at the same time

We have shown that both crucial problems have a common root—the natural contradiction of two basic types of hierarchical abstraction. This root manifests itself also in the way of solution of both above stated problems: in both cases the solution lies in the use of the combination of both abstraction types together.

The general conclusion made from this paper is as follows: generalisation, unlike the aggregation, leads to the need to principally distinguish between the meta-concepts "object" and "concept". This difference is not always primarily visible at the global level (i.e. at the level of the system of objects). Nevertheless, it is well visible at the detail level (i.e. at the level of the object life cycle) especially in two basic situations:

- In the situation where it is a need for expressing the life cycle of the generic concept together with the life cycles of its specific subtypes
- In the situation where it is a need for expressing several different and mutually incompatible life cycles valid for the same class at the same time

This fact can be used as an additional and "physical" reason for distinguishing between the above mentioned types of generic objects. Consequently, it can be used as a proof of the general validity of the need for distinguishing between substantial and temporally dependent types of generalisation.

Acknowledgements The work presented in this paper has been supported by the Czech Science Foundation in the grant project No. P403/10/0303 Enterprise Architecture as Management Principle for SMEs.

References

1. Business Substance Metamodel. http://opensoul.panrepa.org/bsm.html
2. Ebert J, Engels G (1997). Specialization of object life cycle definitions. Fachberichte Informatik 19/95, University of Koblenz-Landau
3. Guizzardi G (2005) Ontological foundations for structural conceptual models. Telematica Instituut Fundamental Research Series No. 15, ISBN 90-75176-81-3, ISSN 1388-1795
4. Heller B, Herre H (2004) Ontological categories in GOL. Axiomathes 14: 71–90 Kluwer Academic
5. Kappel G, Schrefl M (1991) Object/behavior diagrams. In: Proceedings of the 7th international conference on data engineering. IEEE Computer Society, Los Alamitos, pp 530–539

6. Kobryn C (2000) Introduction to UML: structural modelling and use cases. Object modelling with OMG – UML tutorial series. http://www.omg.org
7. Küster JM, Ryndina K, Gall H (2007) Generation of business process models for object life cycle compliance. In: Proceedings of BPM 2007 international conference, LNCS 4714. Springer, Berlin, pp. 165–181
8. Larman C (2002) Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process, 2nd edn. Prentice Hall, Upper Saddle River, NJ. ISBN 0-13-092569-1
9. Repa V (2003) Business system modeling specification. In: Proceedings of the CCCT2003 international conference, IIIS, Orlando, FL
10. Object Management Group (2003) UML OMG unified modeling language specification, v. 1.5. Document ad/03-03-01
11. Object Management Group (2004) UML Superstructure specification, v2.0 document 05-07-04
12. Weisman R (1999) Introduction to UML based SW development process. http://www.softera.com