

On the Classification of Associations

Peter Rittgen

Abstract

When developing an object model, the modeler is confronted with a potentially confusing selection of associations: generalization, aggregation, role and the like. Although domain experts may have some preconceived notion of the meaning of a particular association, these notions do not necessarily coincide with each other or with that of the software engineer. But such ambiguities present a possible source of inadequate modeling and may hence cause severe problems in the design phase. We therefore suggest a classification that reduces the most common associations to only two relation primitives. The consequence of this categorization is a conceptual frontend for object modeling called the concept model. We claim that the domain expert can identify the relation primitives more easily and accurately than the associations themselves. Nevertheless, all commonly known associations map to the set of primitive relation, which enables us to support the migration of a conceptual model to an object model.

Introduction

When developing an object-oriented model, the modeler is confronted with challenging decisions: Which parts of the problem domain are objects and which not? How do the parts relate to each other? Which association applies in a certain case? We claim that the answer to these and similar questions might not be given by the modeler in a straightforward manner. So in order to support him on his way to a “full-blown” object model, we need at least one intermediary step. We suggest that this step consists of designing a conceptual model, or concept model, that functions as a frontend for developing an object model. The Webster dictionary [MeWe 1995] defines concept as

- (1) something conceived in the mind: thought, notion,
- (2) an abstract or generic idea generalized from particular instances.

Whereas the latter definition largely corresponds to that of a class [Mey 1997], the former suggests a broader understanding of the term that may well include the instances themselves. Now, in an object model it is clearly a mistake to confuse class and instance. But in the early phase of analysis and conceptual modeling it might be helpful not to force this distinction in order to facilitate the participation of domain experts and to allow “draft” models. That the weakening of a term can be useful has been recognized even by the formal discipline of mathematics where problems with the consistent axiomatization of set theory led to the introduction of category theory [BaWe 1990]: the stronger but (in certain cases) contradictory concept of a set is replaced by the weaker but more stringent structure of a category.

A substantial amount of different schemata of associations have been suggested so far (e.g. in [HeEd 1994], [Booc 1994] and [RBP+ 1991]). For our purposes we selected the ones that are most commonly named. We also adopt a broad understanding of “association” which includes concepts such as “activity” (generally found in dynamic

models only) and “generalization / specialization” (usually treated separately in the context of inheritance [RBP+ 1991]). So the list of associations consists of:

- generalization/specialization: invoice vs. document
- instance/element: Mr. Miller vs. person
- (de)aggregation: whole vs. part
- role/delegation: person vs. employee
- containment: directory vs. file
- activity/uses: worker vs. workpiece
- unspecified relation: man “is married to” woman

By means of extracting and sharing common characteristics, we can generalize (sub)classes to a (super)class [JCJ+ 1994]. The opposite process of refining a class is called specialization. An instance is a member (element) of a class. Deaggregation refers to the decomposition of a whole (called the aggregate) into its constituents (or parts). Hence the aggregate is made of (or: consists of) the parts. If an object can act in different roles (e.g. the same person can be an employee, a husband and a father), we distinguish between the role player and the role (for details cf. [Fran 1998]). The role player delegates his behaviour when playing the role to this role. Delegation is sometimes seen as dynamic subclassing where the role is a subclass of the player during its enaction.

A containment relation exists between two objects if the one includes, either physically or logically, the other, e.g. a wallet contains money or a list contains entries. Note that there is a principal difference between containment and aggregation: the whole depends on its parts whereas the container does neither depend on the contained objects nor vice versa. A wallet without money is still a wallet, money without a wallet is still money, but a car without a chassis is no longer a car. We would like to point out that there are aggregations that involve containment (car – parts) and such that do not (shareholder – shares [Booc 1994]). An object that performs some activity on, or with the help of, some other object is related to the latter via an “activity/uses” relation. If none of the mentioned association types apply or if we have not yet decided on one, we call the relation unspecified.

However, the people you have to do with in the early phases of analysing and designing an information system are in general not experienced in the use of object-oriented concepts. They will frequently find it hard to select the appropriate association for a specific relation under consideration. Moreover they might not be aware of subtle distinctions such as the one between aggregation and containment. To put it in different words: the association schema is too complex.

In this situation we set out to find the smallest set of common denominators covering all the associations and clustering them into groups of approximately the same size. The result is a classification along two dimensions, namely dynamics and order, yielding four basic relations (section 2). The conceptual model supplied with these relations is enriched incrementally with additional information about the concepts and relations, thus supporting a migration to an object model. In the next step we look into an intuitive notation for these concepts (section 3). Putting everything together, we show an example of a concept model in section 4 and how it is enriched and finally evolves into an object model. Section 5 summarizes the results of this paper and indicates research that remains to be done.

1 The Classification Schema

We conducted a study regarding the classification of associations along the four dimensions

- dynamics: static (S) or dynamic (D),
- order: subordinate (S) or co-ordinate (C),
- level: same (=) or different (\neq),
- scope: broader/narrower (+/-) or different (\neq).

An association between classes A and B is called static if instance objects remain related over time. Otherwise it is dynamic. The instance association itself is static because set membership is so. B is subordinate to A if either B depends on A or A controls B. Otherwise A and B are co-ordinate. Examples for different levels are set vs. element level or system vs. component level. A broader scope includes the elements of the narrower scope.

dimension → association ↓	dynamics	order	level	scope
gener./specialization	S: 94%	S: 89%	=: 62%	+/-: 52%
instance	S: 99%	S: 95%	\neq : 96%	+/-: 93%
aggregation	S: 95%	S: 88%	\neq : 93%	+/-: 52%
unspecified relation	S: 89%	C: 91%	=: 68%	\neq : 54%
role/delegation	D: 94%	S: 98%	\neq : 54%	\neq : 59%
containment	D: 97%	C: 92%	\neq : 53%	\neq : 88%
activity/uses	D: 100%	S: 91%	=: 60%	\neq : 76%

Table 1: Student classification of associations

We asked 187 students to classify associations according to these dimensions during different classes on information systems during the last two years. The questionnaire contained the name of each association together with an example (e.g. “generalization: invoice → document”). The students had little or no practical experience in modeling information systems. Table 1 shows the results (percentages rounded).

Many figures in the columns “level” and “scope” are quite close to 50%. A little decrease in these figures would turn the results into their opposites. Hence these dimensions are not a reliable basis for classification and are dropped. For the remaining dimensions we get the following table:

dimension → association ↓	dynamics	order
gener./specialization	S	S
instance/element	S	S
aggregation	S	S
unspecified relation	S	C
role/delegation	D	S
activity/uses	D	S
containment	D	C

Table 2: Resulting classification schema

Table 2 indicates that “specialization” is perceived as a static, subordinate association: A class A that has been identified as a subclass of B remains so; the meaning of A

depends on that of B . The same holds for “instance” (A being the instance) and “aggregation” (A being the aggregate). In the absence of knowledge concerning dynamics and order, the default assumption for an unspecified relation is static and co-ordinate (i.e. not ordered). The role association is dynamic: one player can act in different roles and a role can be filled with different actors. It is subordinate, too, because the player controls the role. An activity is (obviously) dynamic: the actor controls the object of his activity. Containment is dynamic because objects can be placed into or removed from a container. So a certain contained object is not necessarily in a containment relation all the time. Container and contained object do not depend on each other (as already pointed out above).

2 A Notation for Association Types

Our principal goal is to develop notational elements for the primitives (dynamics and order) rather than for all the associations. The conceptual modeler should not be made to worry about whether a certain relation is of type containment, aggregation or something else. Instead he should worry about the fundamentals:

- Is the association I am about to model consistent over time or not?
- Do the participants in this relation depend on each other or not?

The answers to these questions are then immediately represented graphically in the conceptual model. Later refining information can be added to facilitate the transition to an object model. For the notation, we suggest to adopt the following conventions (see fig. 1):

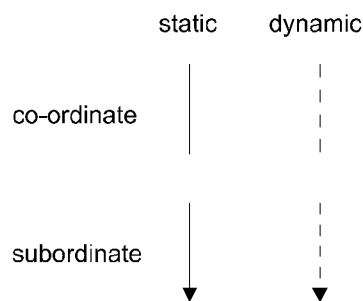


Fig. 1: Notation for the association primitives

Static associations are drawn as solid lines, dynamic relations are represented by a dashed line symbolizing dynamics by the changes between the solid parts and the gaps. Co-ordinated objects are connected with undirected arcs signifying the missing order. Arrows are used to depict subordinate associations. For static relations we draw an arrow from A to B if A depends on B . The same is done for dynamic relations if A controls B . Fig. 2 shows an example of each association in the framework of the conceptual model. The annotations in parentheses only serve to illustrate the example. They are not necessarily part of a conceptual model.

The examples show that there is no one-to-one correspondence between arcs and associations. The solid arrow stands for “generalization” (G), “instance” (I) and “consists of” (C). The dashed arrow denotes activity (A) and role (R) relations. So in order to develop an object model, some additional information is required. To help the designer of the object model, the conceptual relations might be annotated with the capital initial of the respective association.

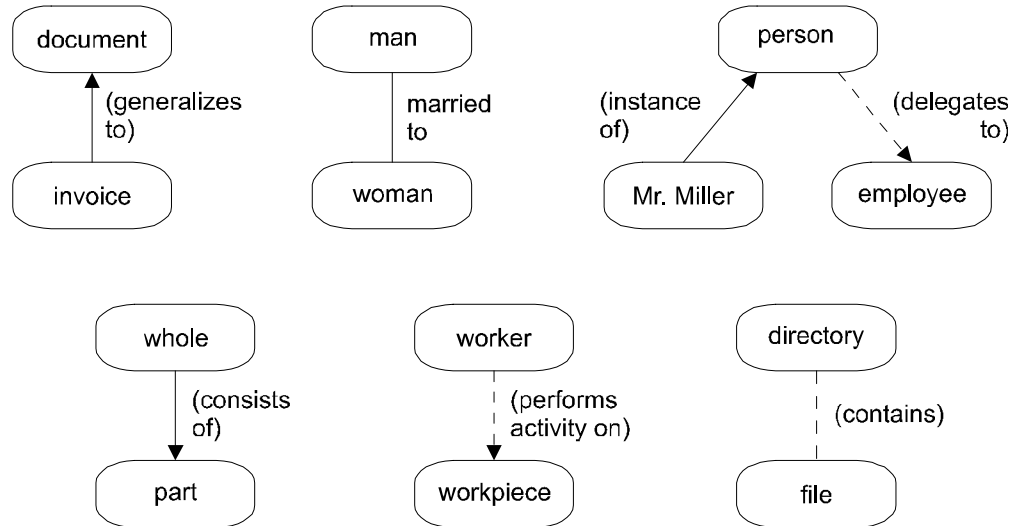


Fig. 2: Examples for associations

3 From Concept Model to Object Model

A concept model is a graph whose nodes are concepts and whose edges are the association primitives identified in section 3. The term “concept” bears close relations to both the term “entity” and the term “object”. It subsumes the terms “class” and “instance”, which are strictly distinguished in object-oriented approaches. In this respect a concept model is weaker than an object model (and hence more conceptual). A concept is drawn as a rounded rectangle. Concepts are related as described in section 3. The arcs can be annotated with the name of the relation. If he wishes the modeler may adorn them also with cardinalities 1 or n . Consider, for example, the concept model shown in fig. 3 (the zig-zag arrow is not part of the model but indicates a questionable relation).

Consultant Johnson is responsible for making a concept for a new planning system in a small production company. First of all, he identifies the people having to do with the PPS system: the customers, the employees and the head of sales Mr. Miller. Hence he creates a concept for each of them and ‘generalizes’ them to the concept “person” (more precisely: he defines the former as static subordinates of the latter). But then he detects a flaw in his reasoning: whereas a customer can be any person (including a juristic person / legal entity), an employee is always a natural person. Moreover, Mr. Miller is not just any person but an employee. All relations described so far are perceived as static by Mr. Johnson because for him a certain person is always either a customer or an employee. He does not see this person in his role as a father, for example. The production unit consists of the manager and a team of welders producing iron gates, which are all employees. Here Johnson immediately realizes that their relation to the concept “employee” is dynamic because he knows that the senior welder acts as production manager when the latter is on holiday. The principal activity of the manager is to create a weekly production plan. This plan indicates the jobs to be performed by the employees. Such a plan is subject to frequent changes, hence the respective relations are marked as dynamic. Johnson inadvertently calls one relation “consists of” because he is not aware of the fact that aggregation is a permanent relation. Had he been urged to draw an object model, this mistake might have been passed unnoticed. But the simpler question whether the relation varies over time will probably be answered correctly.

Hence the relation can be identified as containment in the migration of the concept model to an object model in spite of the mistake.

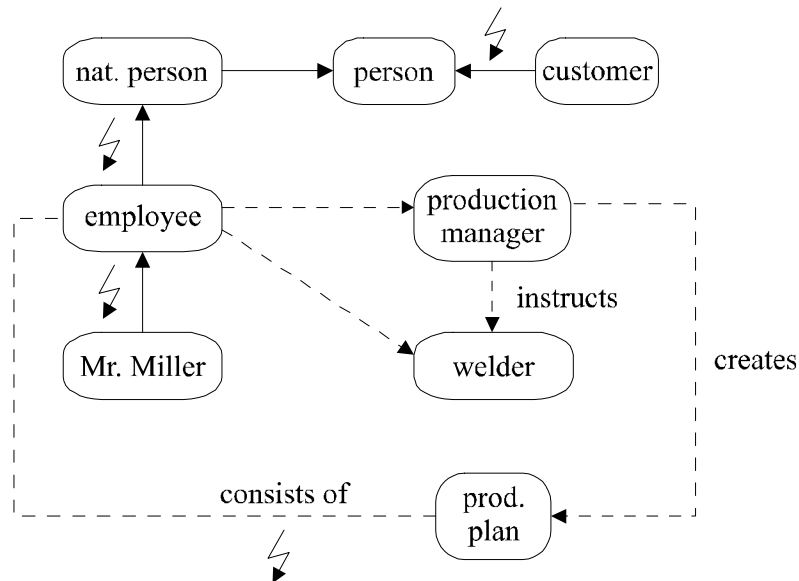


Fig. 3: Example concept model

To transform a concept model into an object model, we perform the following 3 steps:

1. transform concepts into objects,
2. transform conceptual relations into associations,
3. correct “mistakes” in the resulting model.

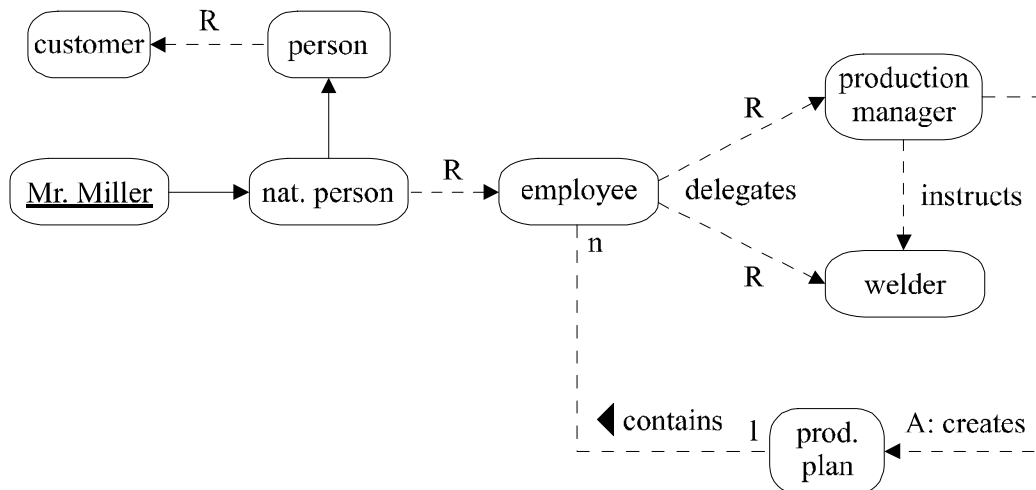


Fig. 4: Object model for the example ($R = \text{Role}$, $A = \text{Activity}$)

Applying this transformation to the concept model of fig. 3 results in the object model of fig. 4. The first step requires the distinction between the type and the element level. Per default we assume a concept to be a class and keep the notation. In the case of

an instance we adopt the UML convention of underlining the name of the object (see Mr. Miller in fig. 4).

In the second step, we add the information about the type of association in the short form of the capital initial as suggested in section 3. Letters “G” and “I” may be omitted so that solid arrows without a distinguishing symbol are taken to be generalization (between two classes) or instance (from instance to class), respectively. As an alternative, the full name of the association may be given (e.g. “contains” in fig. 4). For co-ordinate relations the name (or the symbol) should be accompanied by a little black triangle indicating the ‘reading direction’ (the triangle pointing to the left in fig. 4 means that a production plan contains employees, and not vice versa). For subordinate relations the reading direction is given by the arrowhead. If necessary, missing cardinalities can be supplied.

The third step is the most difficult because it is hard to judge what a “correct” model is. In our example of fig. 3, Mr. Miller, for example, should be an instance of “natural person” and not of “employee”. “employee” and “customer” are rather roles than subclasses of “person” because an employee might well be a customer, too. The inappropriate “consists of” has been replaced already in step 2. After performing these steps, we end up with an object model that still looks very much like the conceptual model we started with. So, in the example, the transition from concept model to object model does not involve complex manipulations. Hence the concept model is a suitable conceptual frontend for developing object models.

4 Conclusion

Our paper claims that object-orientation is essentially a software-engineering paradigm. As a consequence a conceptual modeling language is necessary to foster communication between domain and software experts in a simple and straightforward fashion while at the same time allowing the transition to an object model without substantial reorganization. We suggest the concept model as a candidate. Concepts subsume classes and instances allowing the modeler to abstract from type/element distinctions and to concentrate on the “entities” of his/her problem domain. We classify all the relations between concepts along two simple dimensions: dynamics and order. A graphical notation is developed which encodes these dimensions separately yielding four basic relation types. A three-step procedure transforms concept models into object models making use of the fact that the relation types closely capture the essence of the most common associations.

However, it remains to be shown, e.g. in a comparative field study, that relation primitives can indeed be identified more easily and accurately than associations and that a conceptual model in general and the one suggested in particular really improve the quality of object models.

References

- [Booc 1994] Booch, G.: *Object-oriented analysis and design with applications*. 2nd ed., Benjamin/Cummings, Redwood City CA, 1994.
- [BaWe 1990] Barr, M.; Wells, Ch.: *Category theory for computing science*. Prentice-Hall, New York, 1990.

- [Fran 1998] Frank, U.: *The Memo Object Modelling Language (MEMO-OML)*. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 10, Koblenz, 1998.
- [HeEd 1994] Henderson-Sellers, B.; Edwards, J.M.: *The Working Object, Object-Oriented Software Engineering: Methods and Management*. Prentice-Hall, New York, 1994.
- [JCJ+ 1994] Jacobson, I.; Christerson, M.; Jonsson, P.; Övergård, G.: *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, New York, 1994.
- [Meye 1997] Meyer, B.: *Object-Oriented Software Construction*, Prentice Hall, 1997
- [MeWe 1995] Merriam Webster's Collegiate Dictionary, 10th edition, Merriam Webster, 1995
- [RBP+ 1991] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.: *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs NJ, 1991.