RESEARCH REPORT 0329

SPECIALIZING ASSOCIATIONS

by

M. SNOECK
W. LEMAHIEU

D/2003/2376/29

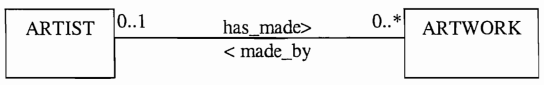
Specializing Associations

M. Snoeck - W. Lemahieu
Katholieke Universiteit Leuven,
Department of Applied Economic Sciences, MIS group
Naamsestraat 69, 3000 Leuven, Belgium
email: {msnoeck, wilfried.lemahieu}@econ.kuleuven.ac.be

Abstract. This paper proposes to apply the concept of specialization to associations. As associations are implemented as properties and subclasses are allowed to redefine inherited properties, refinement of associations occurs de facto at implementation level. It is therefore defendable to allow redefinition of associations at conceptual modeling level as well. This paper proposes a covariant approach to association inheritance and defines the semantics of this concept by means of a set-theoretic approach.

1 MOTIVATION

Inheritance is one of the basic concepts of the object-oriented paradigm. The origin of this paradigm is in programming languages, but very soon, the ideas of the OOP paradigm were adopted by the conceptual modeling community [e.g. 4, 6, 9] and by design and analysis techniques [e.g. 1, 3, 11, 8], resulting in the use of inheritance for conceptual modeling. One of the major differences between conceptual modeling and programming is that whereas the "object type" (also named "entity type" or "class") and the "attribute" in a conceptual model have a direct counterpart in a programming language, namely the class definition and its properties, the concept of "relationship type" or "association" has no direct counterpart in an object-oriented program: the relationship type or association is implemented as a property within a class, resulting in an indirect representation. For example, depending on the directions in which we want to navigate the association, the association "has_made/made_by" in Figure 1 will lead to a property "has_made : SET(ARTWORK)"[1] in class ARTIST and/or a property "made_by: ARTIST" in class ARTWORK.



```
Class ARTIST                          Class ARTWORK
  ...                                   ...
  has_made: SET(ARTWORK)                made_by: ARTIST
  ...                                   ...
end                                   end
```

Figure 1. Example of an association and its implementation

In programming languages, a class inherits the attributes and operations of its supertype and is able a.o. to refine those properties by replacing them with a new version. Inheritance thus applies to the concept of class and to the concept of property: a class can be a refinement of another class and a property can be a refinement of another property. As the concept of "association" is not explicitly present in programming languages, the principle of inheritance has never been directly applied to associations, not in programming languages, but neither in object-oriented analysis or conceptual modeling. However, when a class inherits properties from its parent class, it is allowed to refine the inherited properties by redefining them. Hence, a property that implements an association can be refined as well. In this way *at the programming level*, a specialization class can *de facto* define a specialized version of the associations it is involved in. As class diagrams are intended as specifications of classes in the final code and as it is possible to refine association properties in this code, it would be better if the specialization of associations could already be specified in the class diagram. The need for specialization of associations can easily be illustrated by a case of parallel hierarchies. The conceptual schema in Figure 2 is inspired by a domain model

[1] SET stands for a generic class implementing a collection type (such as e.g. a linked list or an array).

for a company in the telecommunication business. The company sells both material products (like routers and modems) and service products (like web site development and hosting). Although there are some common properties, orders for material products are treated differently from orders for service products. In Figure 2 the general association *orders*$_P$ says that each order orders exactly one product and each product can be ordered in many orders. Products are refined into services and materials. A material order orders materials (only). This is modeled by means of the association *orders*$_M$. Similarly, a service order orders a service, which is modeled by *orders*$_S$.

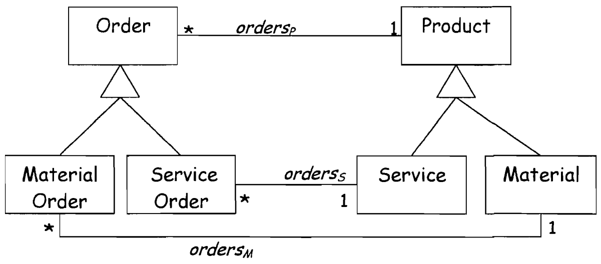


Figure 2. Example of a parallel hierarchy

The current notation does not allow to specify that the associations *orders*$_M$ and *orders*$_S$ are refinements of *orders*$_P$. Strictly speaking, these associations do not replace the definition of the association *orders*$_P$ and hence they exist in parallel. As a result, MATERIAL ORDER has an association *orders*$_M$ with MATERIAL *and, by inheritance,* an association *orders*$_P$ with PRODUCT. Hence a material order is both linked to a material product and to a general product, which can be substituted by either a material product but also by a service product. This is clearly not the intent of the model: the intent of the model is to restrict the ordering of materials and services to material orders and service orders respectively. The example is a typical example of covariance, where the specializations need to narrow the destination of the association. Covariance and narrowing raise particular issues of type safety at the implementation level [7, 10]. In terms of programming by contract [7], the *orders*$_P$ association should be interpreted as "*any kind of* order can be related to one product *of any kind*" and "*any kind of* product can appear in *any kind of* order". According to Shang [10], when subclasses require narrowing, there usually is an untrue promise in the superclass. In the given example, the interpretation of *orders*$_P$ as stated above is indeed an untrue promise. This interpretation is however strongly linked to the use of contracts [7] as a (very useful) technique to describe the capabilities of a class.

The combination of covariance and programming by contract induces a problem of type safety that has not yet found a completely satisfactory solution [7]. The most easy solution is to advocate the use of novariance or contravariance when designing inheritance hierarchies. In a novariant approach, a subclass is not allowed to narrow inherited properties: what has been inherited must be left unchanged. Of course, new features can be added next to the unchanged inherited properties. The contravariant approach follows the principle that a subclass can extend the capabilities of the superclass, also for the inherited properties [15]. The underlying idea is that the subclass can handle more situations and deliver better results. This means that for inherited properties preconditions can be relaxed (but not narrowed) and postconditions can be strengthened (but not relaxed). In [2] this is formulated as a requirement of contravariance for the domains of an association and covariance for the co-domain of an association. This means that an association from A to B can be refined to an association from a subtype of A to a supertype of B. Both in the novariant and in the contravariant approaches, the inherited contracts are fulfilled by the subtype. Although novariance and contravariance are much easier to deal with in terms of ensuring the type safety of programs, covariance is the way people think naturally, especially in a stepwise refinement approach. Also other authors agree that covariance is what we need [7, 10]. This paper defines a covariant approach to inheritance of associations. In doing so, we will not use the interpretation of an association as a "promise" or contract on the capabilities of a class. Rather, the conceptual model is considered as a set of axioms that constrain the set of valid model instances. If we only consider the static dimension of a conceptual model, a model instance can be defined as a set of objects related to each other by means of instances of associations, each object belonging to a class of the model and each association instance belonging to an association (type) defined in the model. The $orders_P$ association now defines that a model instance is valid if for each order instance in class ORDER, the model instance contains an association instance that links this order to a product. The refined associations $orders_S$ and $orders_M$ define constraints that are more stringent than the constraint imposed by the $orders_P$ association. In particular, the $orders_S$ association limits the valid model instances to those instances where each service order instance is linked to a service instance (and likewise for material order instances). As a result, any model instance satisfying the constraints imposed by the $orders_M$ (or $orders_S$) association, also satisfies the constraints of the $orders_P$ association. The interpretation of the association as constraints on a type works fine and is in line with a stepwise refinement approach to modeling: the $orders_P$ association is a *vague, abstract association* that needs to be clarified further in the specialization.

The interpretation of an association as a constraint does not solve the problem completely. In the schema of Figure 2, the $orders_P$ and $orders_M$ /$orders_S$ association exist simultaneously, meaning that a material/service order refers both to a material/service product and by inheritance to a (general) product. This problem (and at the same time the problem of narrowing) can be circumvented by dropping the $orders_P$ association from the schema. However, from a modeling point of view, the association at the generalization level is worth maintaining. One of the primary goals of a conceptual model is communication with end-users and as such, one of the quality criteria of a conceptual modeling language is its ability to create models that are easy to read and understand. In this respect, the $orders_P$ relationship allows to express a general statement that can be refined on a second reading of the model. It also expresses a modeling process of stepwise refinement: taking a global perspective first and then refining the details. The presence of the $orders_P$ association allows to zoom into the details of a model which proves to be a very useful feature, especially in the case of large models. Finally, an additional argument can be given when considering schema evolution. If a new type of product is added in the model of Figure 2, for example "PACKAGE" as shown in Figure 3, then the *orders$_P$* association specifies that also packages can be ordered. If the generalized association is dropped from the model, PACKAGE becomes a kind of dangling product that cannot be ordered, unless a new subclass of order is added, PACKAGE ORDER, in its turn linked to package by means of an new association *orders$_{PA}$*.

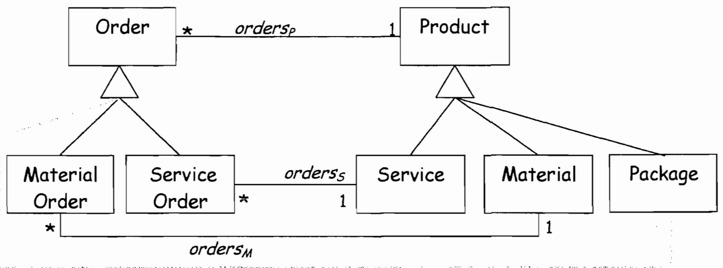


Figure 3. Adding a new type of product.

As we wish to keep the generalized association and the specialized association, we need to introduce a new notation that specifies that the two specialized associations *orders$_M$* and *orders$_S$* are refinements of and hence replace the general association *orders$_P$*. The new notation is depicted in Figure 4. Developing a new notation for specialization of associations is easy. The real challenge is to define the semantics of this notation so as to ensure a precise and unambiguous definition of the semantics. As the semantics of the UML notation are defined in natural language, they are not rigorous enough as a foundation for precise semantics for association specialization. Therefore the next section presents an algebraic (set theoretic) definition of classes, association, and the inheritance relationship between classes. Section 3 then presents the semantics of association specialization. Section 4 discusses association subsetting, a concept very closely related to association specialization.

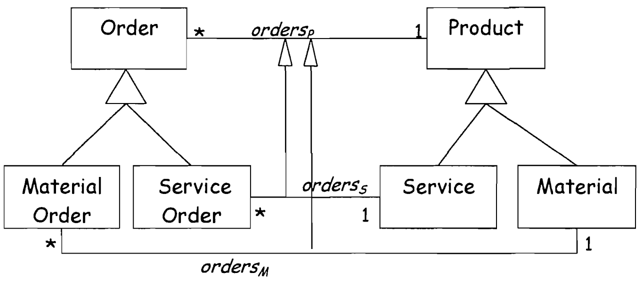


Figure 4. Notation for specialized associations.

2 Algebraic Definition of Classes, Associations and Class Specialization

2.1 Classes and Associations

A conceptual model $\mathcal{M}$ is a pair $(\mathcal{C},\mathcal{R})$, where $\mathcal{C}$ is the set of classes in the model, and $\mathcal{R}$ is the set of associations[2]. A model instance I is a tuple $(O, \mathcal{A})$ where O is the set of object instances and $\mathcal{A}$ is the set of association instances. I is a valid instance of $\mathcal{M}$ if and only if I satisfies all the constraints imposed by $\mathcal{M}$. In that case we write $\mathcal{M} \models I$ (I satisfies $\mathcal{M}$).

Classes are collections of objects. Let O be the set of objects of an instance I. The set of objects O *satisfies* the class definition part $\mathcal{C}$ of $\mathcal{M}$ if the type of every object o in O is a class C in $\mathcal{C}$:

$$\mathcal{C} \models O \Leftrightarrow \forall\, \mathrm{o} \in O: \mathrm{type(o)} \in \mathcal{C}$$

The extent of a class C $\in$ $\mathcal{C}$ is a subset of O denoted by the function instances(C). For the sake of simplicity, we will write C rather than

[2] For the time being, we only consider the static aspects of a model and, for the sake of simplicity, we restrict ourselves to binary relationships and do not consider other attributes than those required to implement the associations.

instances(C) when we refer to the extent of the class. As a result, $C \in \mathcal{C}$ denotes both the type and the extent.

Similarly, every association is a collection of association instances. Since associations aren't necessarily symmetric, each direction of the association is considered separately. An association R is modeled as a link type R_{AB} which is a property of class A and its inverse link type R_{BA} which is property of class B (see Figure 5).

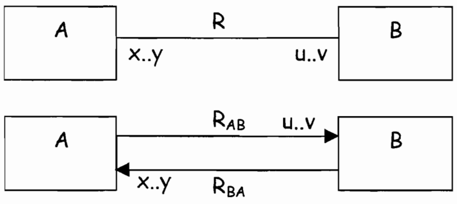


Figure 5. An association and its link types

Definitions

Let $\mathcal{R}$ be the set of associations and $\mathcal{A}$ be the set of association instances.

An association R in $\mathcal{R}$ between the classes A and B from $\mathcal{C}$ is a tuple (R_{AB}, R_{BA}) with

$R_{AB} : A \rightarrow B$, denoting the link type from A to B and

$R_{BA} : B \rightarrow A$, denoting the inverse link type from B to A.

The set of all link types is denoted $\mathcal{L}$: $\mathcal{L} \subseteq 2^{\mathcal{C} \times \mathcal{C}}$, $\mathcal{R} \subseteq \mathcal{L} \times \mathcal{L}$

A is called the domain of the link type R_{AB} and B the co-domain. B is the domain of the link type R_{BA} and A is its co-domain. By definition, R_{AB} and R_{BA} are each other's inverses[3]:

[3] We *define* R_{AB} and R_{BA} as each other's inverses, although in an actual implementation this might not be realized as such.

domain(R_{AB}) = co-domain(R_{BA}) and co-domain(R_{AB}) = domain (R_{BA}) and $(R_{AB})^{-1} = R_{BA}$

The extent of an association $R \in \mathcal{R}$ is considered by taking the instances of the link types separately. However, as two classes might have more than one association between them, a link instance cannot be uniquely identified by the objects it links; we need to know its type as well. As a result, a link instance is defined as a triple (link type, domain object, co-domain object). Hence, $\mathcal{A}$ is a subset[4] of $\mathcal{L} \times \mathcal{O} \times \mathcal{O}$, containing all the link instances. In analogy with classes, R_{AB} is used to denote both the link type and the extent of the link type: instances(R_{AB}) $\subseteq \mathcal{A}$ is written as R_{AB} for short. As to the type of a link instance r, we discern between the *link type* and the *association type* of a link instance. The *link type* (or *l-type* for short) is simply the link type in $\mathcal{L}$, the extent of which contains r. The *association type* (or *a-type* for short) is the association of which the link type of r is part of. Formally:

Let $R = (R_{AB}, R_{BA})$, $r = (R_{AB}, x, y)$[5]

instances (R_{AB}), instances (R_{BA}) $\subseteq \mathcal{A}$

l-type (r) = $R_{AB} \Leftrightarrow r \in R_{AB}$

a-type(r) = $R \in \mathcal{R} \Leftrightarrow [\, r \in R_{AB}$ or $r \in R_{BA}]$

instances(R) = instances(R_{AB}) $\cup$ instances (R_{BA})

[4] Notice that the set of link instances is defined as a set and not as a bag. As a consequence, each occurrence in a link is uniquely identified by the name of the link type and the identifications of the participating objects. In case the same objects can be related more than once by means of the same association, the association should be modelled as a class.

[5] When no confusion is possible, we will write r = (x,y) as a shorthand for r = (R_{AB},x,y)

Associations can be used to navigate from one class to another. The set of objects that can be reached from a given object a by navigating along a link type R_{AB} can be defined as the image of an object and is denoted as $a\bullet R_{AB}$.

$$\forall a \in A: a\bullet R_{AB} = \{b \in B \mid (R_{AB},a,b) \in R_{AB}\} \subseteq B$$

Each link type has a minimum and maximum cardinality constraint. The cardinality of R_{AB} tells how many $b \in B$ can be connected to an $a \in A$. A minimum cardinality of 0 means that the link type is optional, which means there is no constraint on the minimum number of association instances an object a of A must participate in. A minimum cardinality of 1 means that the link type is mandatory: Every a in A must be connected to a b in B. A maximum cardinality of '*' means that there is no constraint on the number of association instances any a of A can be involved in. A maximum cardinality of '1' means that any a of A can be connected to at most 1 b of B. In the class diagram, the cardinality of an association is denoted as an interval [minimum, maximum] next to the co-domain of the link type. The four different possibilities are:

	minimum constraint ?	maximum constraint ?
R_{AB} is [0,*]	no	no
R_{AB} is [0,1]	no	yes
R_{AB} is [1,*]	yes	no
R_{AB} is [1,1]	yes	yes

Formally, this can be formulated as follows:

Every link type has a minimum and maximum cardinality:

$\forall R_{AB} \in \mathcal{L}: \min(R_{AB}) \in \{0, 1\}$ and

$\forall R_{AB} \in \mathcal{L}: \max(R_{AB}) \in \{1, *\}$

$$\min(R_{AB}) = 1 \quad \Rightarrow \forall\, a \in A: \#\,(a \bullet R_{AB}) \geq 1$$
$$\Rightarrow \forall\, a \in A: \exists\, b \in B: (R_{AB}, a, b) \in R_{AB}$$

$$\max(R_{AB}) = 1 \quad \Rightarrow \forall\, a \in A : \#\,(a \bullet R_{AB}) \leq 1$$
$$\Rightarrow \forall\, (R_{AB}, a, b) \in R_{AB}: \neg\, [\exists\, (R_{AB}, a, c) \in R_{AB},\, c \neq b\,]$$
$$\Rightarrow \forall\, (R_{AB}, a, b) \in R_{AB}: (R_{AB}, a, c) \in R_{AB} \Rightarrow b = c$$

Notice that the cardinalities are *constraints*, which means that an actual instance of the model can "by accident" be more stringent than required. For example, if R_{AB} is modeled as [0,1], we might have an instance of the model where every a from A is connected to a $b \in B$, resulting in R_{AB} behaving (temporarily) as though it were [1,1].

In order to satisfy the model $\mathcal{M}$, every link instance r in $\mathcal{A}$ should have a type R in $\mathcal{R}$. In addition, the set of instances should satisfy the cardinality constraints. These can be formulated in terms of the image of an object along a link type. Let $R = (R_{AB}, R_{BA}) \in \mathcal{R}$, then

$$\mathcal{R} \models \mathcal{A} \Leftrightarrow [\, (\forall\, r \in \mathcal{A} : \text{a-type}\,(r) \in \mathcal{R}) \wedge (\forall\, R \in \mathcal{R}: R \models \mathcal{A}\,)\,]$$

$$\begin{aligned} R \models \mathcal{A} \Leftrightarrow \quad & \min(R_{AB}) = 1 \Rightarrow \forall\, a \in A : \#\,(a \bullet R_{AB}) \geq 1 \\ \text{and} \quad & \min(R_{BA}) = 1 \Rightarrow \forall\, b \in B : \#\,(b \bullet R_{BA}) \geq 1 \\ \text{and} \quad & \max(R_{AB}) = 1 \Rightarrow \forall\, a \in A : \#\,(a \bullet R_{AB}) \leq 1 \\ \text{and} \quad & \max(R_{BA}) = 1 \Rightarrow \forall\, b \in B : \#\,(b \bullet R_{BA}) \leq 1 \end{aligned}$$

Figure 6 shows an example of a model and a valid instance of this model. Every double arrow connecting an artist instance to an artwork instance represents the two links (artist, artwork) and (artwork, artist) which are occurrences of the link types *has_made* and *made_by* respectively. The model instance satisfies the only cardinality constraint in the model: every artwork is related to at most one artist.

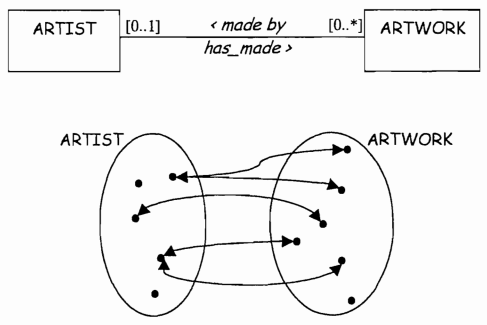


Figure 6. Model with sample instance

2.2 Specialization of Classes

The following definitions formalize the structural aspects of generalization/specialization:

The ISA_c or $<_c$ relationship is a partial order between classes. When S ISA_c G (G $<_c$ S), we say that S is the specialization and G is the generalization class[6]. The ISA_c hierarchy must satisfy the following restrictions:

1) Object types have at most one generalization type (we do not yet allow multiple inheritance).
2) The ISA_c hierarchy must be acyclic.

The ancestor relationship between classes is denoted as $<_c^+$ and is the reflexive transitive closure of $<_c$. The classes that are directly or indirectly a generalization of a class S, are called the *ancestor classes* of S. The *topmost ancestor* of a class S is that ancestor that is on the top of the generalization hierarchy. The ancestor that is a *direct generalization* of a class S is called the *parent* of S. Classes that are directly or indirectly a specialization of class G, are called the *descendants* of G. The classes that are *direct specializations* are called the *children* of G. A class G is both an ancestor and a descendant of

[6] The direction of the '<' sign might seem somewhat contra-intuitive, but is motivated by the fact that the specialisation is considered as an extension of the generalisation.

itself. The *extent* of a class P is the set of all occurrences of this class. The *deep extent* is the union of extents of descendant classes of P, including the extent of P.

Formal definition

$<_c^+$ is a hierarchical partial order on $\mathcal{C}$

$\Leftrightarrow$ (1) $\forall\, G \in \mathcal{C}: G <_c^+ G$

(2) $\forall\, G, S \in \mathcal{C}: G <_c^+ S$ and $S <_c^+ G \Rightarrow G = S$

(3) $\forall\, G, S, T \in \mathcal{C}: G <_c^+ S$ and $S <_c^+ T \Rightarrow G <_c^+ T$

(4) $<_c$ is the cover of $<_c^+$:

$\forall\, G, S \in \mathcal{C}, G <_c S \Rightarrow G <_c^+ S$

$\forall\, G, S \in \mathcal{C}, G \neq S$:

$G <_c^+ S$ and $\neg\,(\exists\, H \in \mathcal{C}: G <_c^+ H$ and $H <_c^+ S) \Rightarrow G <_c S$

(5) $<_c$ must be a hierarchy:

$\forall\, G, G', S \in \mathcal{C}: G <_c S$ and $G' <_c S \Rightarrow G = G'$

Let $G, S \in \mathcal{C}$. If $G <_c S$ then we say that G is a generalization or parent of S and S is a specialization or child of G.

$\gamma_+(P) = \{G \in \mathcal{C} \mid G <_c^+ P\}$ *(ancestors of P)*

$\gamma_-(P) = \{S \in \mathcal{C} \mid P <_c^+ S\}$ *(descendants of P)*

$\gamma_{max}(P) = \{G \in \mathcal{C} \mid G <_c^+ P$ and $\neg\,\exists\, H \in \mathcal{C}: H <_c G\}$ *(topmost ancestor)*

$.^+$ is a postfix operator that takes the deep extent of an object type and is defined as: $P^+ = \cup\,\{S \mid S \in \gamma_-(P)\}$

In a strong typing approach, it is required that each object belongs to exactly one and always the same class for the whole time of its existence. As a result, the generalization and specialization types have disjoint extents. For the same reason, overlapping subclasses are not allowed and objects are not allowed to migrate between subclasses or between subclass and superclass. However, the superset/subset hierarchy can be reconstructed by working with the deep extent of object types (see Figure 7): the deep extent of a specialized class is always a subset of the deep extent of its generalization:

$$\forall\ S, G: G <_C S \Rightarrow S^+ \subseteq G^+$$

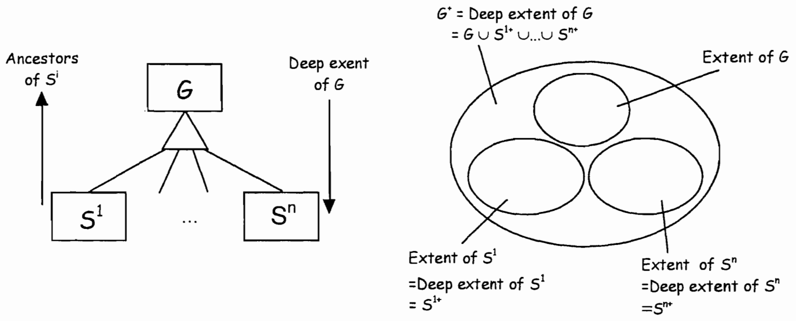


Figure 7. Extent/deep extent and superset/subset hierarchy

2.3 Associations over Specialized Classes

The introduction of specialization for classes, requires a redefinition of associations. Indeed, when a supertype is involved in an association, all subtypes inherit the association from their supertype. As a result, the property R_{AB} now also applies to objects in each subtype A' of A. The object that is linked by R_{AB} to an object a from A, can be an object b from B itself but also from any descendant class of B. The fact that an object a can also be from a descendant type of B rather than from B only is called the principle of "substitution": whenever an object from a given class is required, it can always be substituted by an object of a specialized class. Similarly, the property R_{BA} also applies to each object b' in a subtype B' of B and its co-domain is A^+. Hence, a link type R_{AB} with domain A and co-domain B, encompasses tuples (a',b') with a' element of a descendant A' of A and b' element of a descendant B' of B. In other words, the domain of a link type R_{AB} is the deep extent of A and its co-domain is the deep extent of B. The definition of relationships and links is modified accordingly:

Definitions

An association R in $\mathcal{R}$ between the classes A and B from C is a tuple (R_{AB}, R_{BA}) with

$R_{AB} : A^+ \rightarrow B^+$, denoting the link type from A^+ to B^+ and

$R_{BA} : B^+ \rightarrow A^+$, denoting the inverse link type from B^+ to A^+.

A^+ is the domain of the link type R_{AB} and B^+ the co-domain. B^+ is the domain of the link type R_{BA} and A^+ is its co-domain. By definition, R_{AB} and R_{BA} are each other's inverses:

domain(R_{AB}) = co-domain(R_{BA}) = A^+ and co-domain(R_{AB}) = domain (R_{BA}) = B^+ and $(R_{AB})^{-1} = R_{BA}$

When using a relationship R_{AB} to navigate from A to B, the set of objects that can be reached from a given $a \in A^+$ is a subset of the deep extent of B: $\forall\, a \in A^+ : a \bullet R_{AB} \subseteq B^+$ and similarly: $\forall\, b \in B^+ : b \bullet R_{BA} \in A^+$.

The formulation of maximum and minimum cardinalities remains the same, except that it now applies to all elements of the deep extent of A and B:

$$
\begin{aligned}
R \models \mathcal{A} \Leftrightarrow \quad & \min(R_{AB}) = 1 \Rightarrow \forall\, a \in A^+ : \#(a \bullet R_{AB}) \geq 1 \\
\text{and} \quad & \min(R_{BA}) = 1 \Rightarrow \forall\, b \in B^+ : \#(b \bullet R_{BA}) \geq 1 \\
\text{and} \quad & \max(R_{AB}) = 1 \Rightarrow \forall\, a \in A^+ : \#(a \bullet R_{AB}) \leq 1 \\
\text{and} \quad & \max(R_{BA}) = 1 \Rightarrow \forall\, b \in B^+ : \#(b \bullet R_{BA}) \leq 1
\end{aligned}
$$

3 Specialization of Associations

As argued in section 1, there are good reasons to allow applying the concepts of generalization and specialization to associations. This section formalizes the generalization and specialization of associations. An association R between A and B is composed of two link types: R_{AB}, which is a property of class A and R_{BA}, which is a property of class B. Specialization classes of both A and B can independently refine or not refine the inherited properties. For a link type property, both the co-domain and the cardinality constraints are subject to refinement. As a result, several cases can be considered. In the first case, the co-domain classes of the inherited association are refined symmetrically by descendants of A and B, possibly complemented with cardinality refinement. In the second case, only one of the two link types is refined. Finally, the third case discusses the case of asymmetric specialization without narrowing of the destination: only the cardinality constraints are refined.

Before treating the different cases, association inheritance is defined.

Definition

The ISA_a or $<_a$ relationship is a partial order between associations. When R' ISA_a R (R $<_a$ R'), we say that R' is the specialized association and R is the generalized association. The ISA_a hierarchy must satisfy the following restrictions:

> 1) The domain and co-domain of the link types composing R' must be descendants of the domain and co-domains of the composing link types of R (which includes R itself).[7]
>
> 2) The specialized link type inherits and is allowed to narrow the cardinality constraints of the parent link type.
>
> 3) Associations have at most one generalization type (we do not allow multiple inheritance).
>
> 4) The ISA_a hierarchy must be acyclic.

[7] Notice that in a contravariant approach, it is not allowed to narrow both the domain and co-domain of an association. Rather, the contravariant approach advocates to narrow the domain co-variantly and to broaden the co-domain contravariantly. Hence, in a contravariant approach, when a link $R'_{A'B'}$ is a subtype of a link R_{AB}, then B' should be a supertype of B. Castagna [2] therefore suggests the name co-contravariance.

Formal definition

Let A' be a subtype of A and B' be a subtype of B. Then

$\forall$ R, R' $\in \mathcal{R}$:

(0)$R <_a^+ R' \Rightarrow R_{AB} <_a^+ R'_{A'B'}$ and $R_{BA} <_a^+ R'_{B'A'}$

(1)$R_{AB} <_a^+ R'_{A'B'} \Rightarrow A <_c^+ A'$ and $B <_c^+ B'$

(2)$R_{AB} <_a^+ R'_{A'B'} \Rightarrow [\ \min(R_{AB}) = 1 \Rightarrow \min\ (R'_{A'B'}) = 1] \wedge [\max(R_{AB}) = 1 \Rightarrow \max\ (R'_{A'B'}) = 1]$

(3)$<_a^+$ is a hierarchical partial order on $\mathcal{R}$

$\Leftrightarrow$ (a) $\forall R \in \mathcal{R}: R <_a^+ R$

(b) $\forall R, R' \in \mathcal{R}: R <_a^+ R'$ and $R' <_a^+ R \Rightarrow R = R'$

(c) $\forall R, R', R'' \in \mathcal{R}$: $R <_a^+ R'$ and $R' <_a^+ R'' \Rightarrow R <_a^+ R''$

$<_a$ is the cover of $<_a^+$. $<_a^+$ can be recovered by taking the reflexive transitive closure of $<_a$. $<_a$ must be a hierarchy:

(4) $\forall R, R', R'' \in \mathcal{R}$: $R <_a R''$ and $R' <_a R'' \Rightarrow R = R'$

Let R, R' $\in \mathcal{R}$. If $R <_a R'$ then we say that R is a generalization of R' and R' is a specialization of R.

$\gamma_+(R) = \{G \in \mathcal{R} \mid G <_a^+ R\}$ *(Ancestors of R)*

$\gamma_-(R) = \{S \in \mathcal{R} \mid R <_a^+ S\}$ *(descendants of R)*

$\gamma_{max}(R) = \{G \in \mathcal{R} \mid G <_a^+ R \text{ and } \neg \exists H \in \mathcal{R}: H <_a G\}$ *(topmost ancestor)*

$.^+$ is a postfix operator that takes the deep extent of an association and is defined as:

$R^+ = \cup \{S \mid S \in \gamma_-(R)\}$

The main problem that remains to be solved is to determine the type of an association instance. We will define this for each of the following cases separately.

3.1 Symmetric refinement

The most obvious case is the one with parallel hierarchies such as in the example of Figure 2. Figure 8 represents an abstract version of the same type of hierarchy. In this case both the domain A and the co-domain B of the association R have been specialized into respectively A' and B'. The association R has been refined to an association R' that links the specialized classes A' and B'. In practice this means that A' inherits the link type R_{AB} from A, but overrides it with $R'_{A'B'}$. Similarly for B', which overrides R_{BA} with $R'_{B'A'}$.

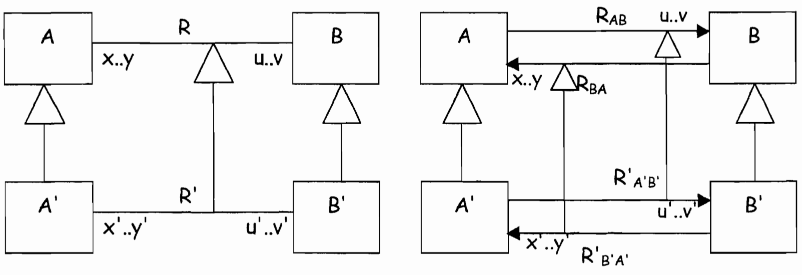


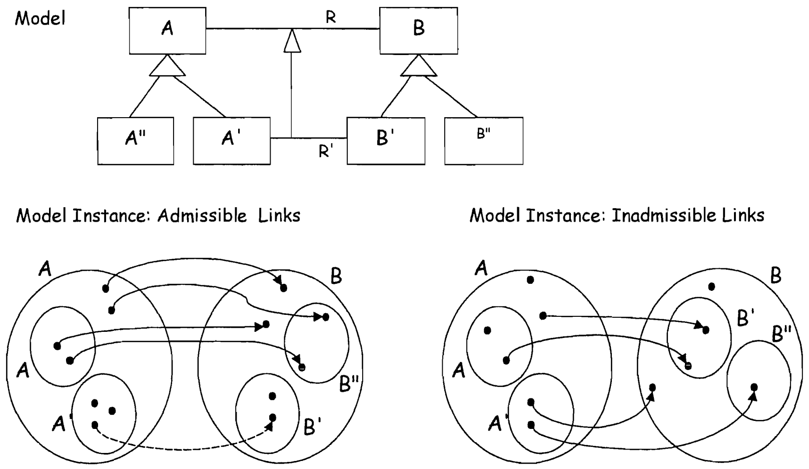
Figure 8. Symmetric refinement

Semantics questions that need to be answered are

1) What are the admissible link instances between elements of A^+ and elements of B^+ ?

2) What is the type of a link instance ?

Let us consider the model in Figure 9. Link instances between elements of A ∪ A" and elements of B ∪ B" are in R. And link instances between elements of A' and B' are in R'. But what about a link instance between an element a' in A' with an element b in B. As B defines the co-domain of R_{BA} as A^+, the link instance is admissible from the point of view of B and hence, (b, a') would be an element of R_{BA}. It would then by definition be an element of R_{AB}, since R_{AB} and R_{BA} are by definition each other's inverses. However, because A' overrides the link type R_{AB} with $R'_{A'B'}$, it should be an instance of $R'_{A'B'}$. But the redefinition of $R'_{A'B'}$ specifies that the co-domain of $R'_{A'B'}$ is B'. Hence, from the point of view of A', a link (a',b) is not admissible. A similar reasoning applies to a link instance (a,b') with $a \in A$ and $b' \in B'$. The lower part of Figure 9 shows the admissible and inadmissible link instances.



Model
A
R
B
A''
A'
R'
B'
B''
Model Instance: Admissible Links
Model Instance: Inadmissible Links
A
B
A
A'
B''
B'
A
B
B'
B''
A
A'

Figure 9. Admissible and inadmissible link instances in parallel hierarchies. The dashed link instance is in R', the solid link instances are in R. Notice that there is no refined link type between A'' and B''

As a result, the domain of R_{AB} is $A^{+}\backslash A'$ and its co-domain is $B^{+}\backslash B'$

Definition

domain(R_{AB}) = $A^{+}\backslash \{A'^{+} \mid A <_c A' \text{ and } \exists R' \in \mathcal{R}, R <_a R' \text{ and domain}(R') = A'^{+}\}$

co-domain(R_{AB}) = $B^{+}\backslash \{B'^{+} \mid B <_c B' \text{ and } \exists R' \in \mathcal{R}, R <_a R' \text{ and co-domain}(R') = B'^{+}\}$

When a link type $R'_{A'B'}$ is a specialization of a link type R_{AB}, it inherits the cardinality constraints of R_{AB}. In a covariant approach, $R'_{A'B'}$ can only add constraints, but must respect the inherited constraints. Hence, $R'_{A'B'}$ must keep the minimum constraint and can add one in case R_{AB} had no minimum constraint. A similar reasoning applies to the maximum constraint. The possibilities for the cardinalities of the descendant are given in Table 1[8].

Table 1:
Meta-constraints on cardinality constraints of specialized link types.

Generalized link type	minimum constraint ?	maximum constraint ?	possible cardinalities for specialized link type
R_{AB} is (0,*)	no	no	(0,*) (0,1), (1,*), (1,1)
R_{AB} is (0,1)	no	yes	(0,1), (1,1)
R_{AB} is (1,*)	yes	no	(1,*), (1,1)
R_{AB} is (1,1)	yes	yes	(1,1)

This table can easily be extended to a more general form of expressing cardinalities. If the cardinalities of a link type are expressed as intervals of natural numbers, then the specialization link type is allowed to raise the lower bound and decrease the upper bound. Formally:

Let the cardinality of R_{AB} be expressed as R_{AB} is [min, max] with min, max $\in \mathbb{N}$

then $R_{AB} <_a R'_{A'B'}$ and $R'_{A'B'}$ is [min', max'] $\Rightarrow$ min $\leq$ min' and max $\geq$ max'

Notice that this even allows a descendant class to disinherit an optional link type by refining a cardinality of [0,n] to [0,0].

[8] Notice that Table 1 identifies meta-constraints. Whereas constraints are elements in the model that put constraints on the validity of model instances, meta-constraints are elements in the meta-model that put constraints on the model itself. Meta-constraints are pertinent to schema evolution, as they determine the validity of newly added subclass definitions.

3.2 Asymmetric specialization

A less obvious case is were only one class specializes the association. We discuss the case where the subtype A' refines the co-domain of R_{AB} to B', but the subtype B' does not refine the co-domain of R_{BA}.

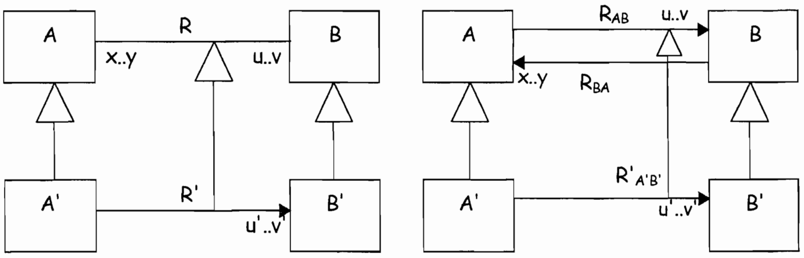


Figure 10. Asymmetric specialization.

In this case, a link instance (a,b) is an element of R_{AB}, and its reverse link instance (b,a) is an element of R_{BA} ($a \in A$, $b \in B$). A link instance (a',b) with $a' \in A'$ and $b \in B$, and its inverse link (b,a') are inadmissible, since A' imposes the constraint that elements of A' should be linked to elements of B'. A link (b',a) and its inverse link (a, b') are however allowed, since B' does not refine R_{BA}. These links are respectively elements of R_{BA} and R_{AB}. A link (a',b') , with $a' \in A'$ and $b' \in B'$ is an element of $R'_{A'B'}$. The inverse link type of $R'_{A'B'}$, namely $R'_{B'A'} = (R'_{A'B'})^{-1}$ is only implicitly defined and a subset of R_{BA}, namely the projection of R_{BA} on (B', A') :

$$(R'_{A'B'})^{-1} = R'_{B'A'} = R_{BA}/(B', A') = \{(b,a) \in R_{BA} \mid b \in B', \text{and } a' \in A'\}$$

In addition to refining the co-domain, $R'_{A'B'}$ can also impose additional cardinality constraints in the same way as for a symmetric specialization (see Table 1).

Figure 11 shows an example of an asymmetric specialization. The generalized association specifies that an artist can make zero to many artworks and that an artwork is made by 0 to one artist (zero means the artist is unknown). PAINTER is a subclass of ARTIST and PAINTING is a subclass of ARTWORK. The *has_made* link type is refined and renamed to *has_painted.* The co-domain of *has_painted* is narrowed to paintings and a cardinality constraint is added, stating that a painter has painted at least one painting. In the other direction, the *made_by* link type is not refined as a painting can also be produced by another type of artist (e.g. a sculptor occasionally producing a painting). Notice that this model defines that a painter cannot be linked to other artwork than paintings since the *has_painted* link type replaces the *has_made* link type. Most likely however, a painter can also produce other artwork than paintings. This situation can be expressed by using relationship *subsetting* rather than specialization, as discussed in section 4.

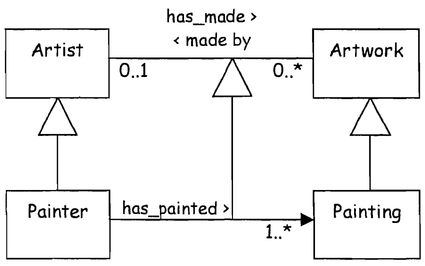


Figure 11. Example of an asymmetric specialization

3.3 Asymmetric specialization without narrowing of the destination

In the previous cases, the refined link type narrows the co-domain to a subtype of the original co-domain. A link type can however also be refined without narrowing the co-domain, e.g. only by specifying an additional cardinality constraint.

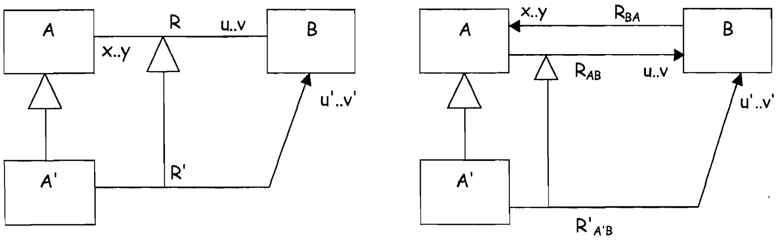


Figure 12. Asymmetric specialization without narrowing

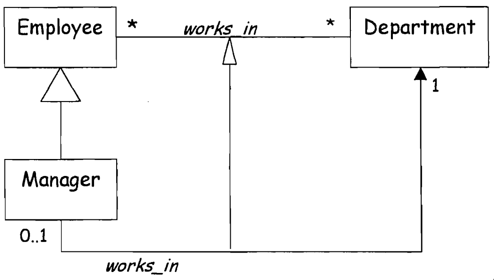


Figure 13.
Example of an asymmetric association, without narrowing of co-domain

Figure 13 gives and example of such a specialization. As a general rule, employees work in zero to many departments: the link type is optional. The subclass MANAGER refines the link type *works_in* to a mandatory link type with in addition a multiplicity constraint of one: each manager works in exactly 1 department. The meta-constraints that apply to such refinement are the same as those for previous cases and are given in Table 1.

4 Association Subsetting

Defining a link type as a subset of another link type is closely related to link type specialization. The major difference is that the subsetting link type does not replace the superset link type but exists next to it. In contrast, a specialized link type replaces its generalized link type. The major similarity is that the deep extent of a specialized link type is a subset of the deep extent of the generalized link type.

This section illustrates the concept of link type subsetting by means of a few examples, without defining it formally, as this is beyond the scope of this paper. The major goal of this section is to contrast link type subsetting with link type specialization. Association subsetting is an already existing concept: it is for example defined in NIAM [16] and ORM [5].

4.1 Subsetting of associations between specialized classes

Suppose that in Figure 11 a painter can also make other artwork than paintings, but that *has_painted* links an artist to his/her paintings whereas *has_made* links the artist to all his/her artwork. In this case, the *has_painted* link type defines a subset of the *has_made* link type. This can be modeled by using association subsetting rather than subtyping, as shown in Figure 14.

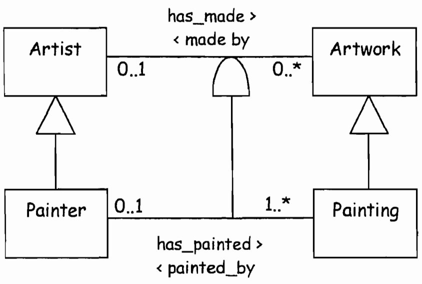


Figure 14. Example of association subsetting

A link instance between a painter and one of his/her paintings (e.g. (Monet, Impression)) is now both an element of the *has_made* link type and of the *has_painted* link type[9].

A link type that is defined as a subset of another link type, does not replace the latter. As a result, the painter class has both the properties *has_made* and

[9] As a result, link type subsetting, does not allow for strong typing of link instances. As long as they are not treated as objects in an implementation, this will not yield problems.

has_painted. With link type subsetting, we specify that the paintings that are linked through the has_painted link type with a particular painter, are also linked to this painter whenever we consider all the artwork made by this painter. More formally, link type subsetting implies that Monet•has_painted $\subseteq$ Monet•has_made.

As to the cardinality constraints, the meta-model constraints are different from those applying to link type specialization. When the *has_made* link type is mandatory, the *has_painted* link type needs not to be mandatory as well: if it is required for an artist to have produced at least one artwork, this artwork needs not to be a painting, not even for a painter. Hence, a minimum cardinality constraint of one can be relaxed by the subsetting link type. However, a maximum cardinality constraint of one must be respected by the subsetting link type: if an artwork can be produced by at most one artist, the same must holds for a painting. Hence, a maximum cardinality constraint of one cannot be relaxed by the subsetting link type. The meta-constraints are summarized in Table 2.

Table 2: Constraints for subsetting link type $R'_{A'B'}$

Superset link type	minimum constraint ?	maximum constraint ?	possible cardinalities for specialized link type $R'_{A'B'}$
R_{AB} is (0,*)	no	no	(0,*) (0,1), (1,*), (1,1)
R_{AB} is (0,1)	no	yes	(0,1), (1,1)
R_{AB} is (1,*)	yes	no	(0,*) (0,1), (1,*), (1,1)
R_{AB} is (1,1)	yes	yes	(0,1), (1,1)

4.2 Asymmetric Subsetting

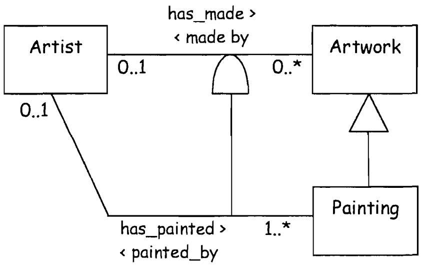


Figure 15. Example of asymmetric link subsetting

In Figure 15, only the class ARTWORK is refined. The model now defines that *has_painted* links artists to paintings and is a subset of the *has_made* link type. Again, *has_painted* does not replace *has_made* and hence the class ARTIST has both types of links. It should be noted that since the domain of the subsetting link type is equal to the domain of the superset link type, a minimum cardinality of one for the subsetting link type implies a minimum cardinality of one for the superset link type. In the given example, any artist will always have at least one artwork, because (s)he must have at least one painting. Strictly speaking, this does not violate the optional nature of the *has_made* link, but for the clarity of the model, such situation is to be avoided. The adapted set of meta-constraints is shown in Table 3.

Table 3: Meta-Constraints for link subsetting when A' = A

Superset link	minimum constraint ?	maximum constraint ?	possible cardinalities for specialized link $R'_{AB'}$
R_{AB} is (0,*)	no	no	(0,*) (0,1)
R_{AB} is (0,1)	no	yes	(0,1)
R_{AB} is (1,*)	yes	no	(0,*) (0,1), (1,*), (1,1)
R_{AB} is (1,1)	yes	yes	(0,1), (1,1)

4.3 Subsetting of an association with the same participants

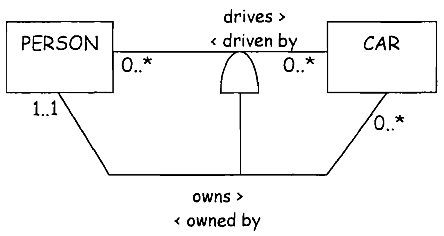


Figure 16 Link subsetting between the same classes.

As a final example, let us consider association subsetting between two identical classes: none of the participating classes has been specialized. In the example of Figure 16, the *owner* association between PERSON and CAR is a subset of the *driver* association. Hence, the owner is always considered to be a driver. As a result, making the *owned_by* link type mandatory implies that the *driven_by* link type should be mandatory as well. The meta-constraints that apply here are the same as in Table 3.

Notice that this type of situation cannot occur for association specialization: whenever an association R' between A' and B' is a specialization of an association R between A and B, at least A and A' or B and B' must be distinct classes.

5 Model satisfaction with Association Inheritance

With the definitions given in sections 3 and 2, we can ensure that an Instance I satisfying a model $\mathcal{M}'$ where an association R has been refined to R' also satisfies the same model $\mathcal{M}$ without refinement for R.

Theorem

Assume that $\mathcal{M} = (C, \mathcal{R})$ and $\mathcal{M}' = (C', \mathcal{R}')$

with $C' = C$, $R \in \mathcal{R}$

and $\mathcal{R}' = \mathcal{R} \cup \{R'\}$, $R' = (R'_{A'B'}, R'_{B'A'})$, $R <_a R'$

Then

$$\mathcal{M}' \models I \Rightarrow \mathcal{M} \models I$$

Proof:

$\mathcal{M}' \models I$

$\Rightarrow \mathcal{C}' \models I$ and $\mathcal{R}' \models I$

$\Rightarrow \mathcal{C}' \models I$ and $\forall \mathrm{T} \in \mathcal{R}' : \mathrm{T} \models I$

$\Rightarrow \mathcal{C}' \models I$ and $\mathrm{R}' \models I$ and $\mathrm{R} \models I$ and $\forall \mathrm{T} \in \mathcal{R}'/\{\mathrm{R}', \mathrm{R}\} : \mathrm{T} \models I$

$\Rightarrow \mathcal{C}' \models I$ and $\forall \mathrm{T} \in \mathcal{R}'/\{\mathrm{R}', \mathrm{R}\} : \mathrm{T} \models I$

and $\min(\mathrm{R}'_{A'B'}) = 1 \Rightarrow \forall a \in A'^{+} : \#(a \bullet \mathrm{R}'_{A'B'}) \geq 1$

and $\min(\mathrm{R}'_{B'A'}) = 1 \Rightarrow \forall b \in B'^{+} : \#(b \bullet \mathrm{R}'_{B'A'}) \geq 1$

and $\max(\mathrm{R}'_{A'B'}) = 1 \Rightarrow \forall a \in A'^{+} : \#(a \bullet \mathrm{R}'_{A'B'}) \leq 1$

and $\max(\mathrm{R}'_{B'A'}) = 1 \Rightarrow \forall b \in B'^{+} : \#(b \bullet \mathrm{R}'_{B'A'}) \leq 1$

and $\min(\mathrm{R}_{AB}) = 1 \Rightarrow \forall a \in A^{+} \backslash A' : \#(a \bullet \mathrm{R}_{AB}) \geq 1$

and $\min(\mathrm{R}_{BA}) = 1 \Rightarrow \forall b \in B^{+} \backslash B' : \#(b \bullet \mathrm{R}_{BA}) \geq 1$

and $\max(\mathrm{R}_{AB}) = 1 \Rightarrow \forall a \in A^{+} \backslash A' : \#(a \bullet \mathrm{R}_{AB}) \leq 1$

and $\max(\mathrm{R}_{BA}) = 1 \Rightarrow \forall b \in B^{+} \backslash B' : \#(b \bullet \mathrm{R}_{BA}) \leq 1$

$\Rightarrow$[10] $\mathcal{C}' \models I$ and $\forall \mathrm{T} \in \mathcal{R}'/\{\mathrm{R}', \mathrm{R}\} : \mathrm{T} \models I$

and $\min(\mathrm{R}_{AB}) = 1 \Rightarrow \forall a \in A^{+} \backslash A' : \#(a \bullet \mathrm{R}'_{A'B'}) \geq 1$
and $\forall a \in A'^{+} : \#(a \bullet \mathrm{R}'_{A'B'}) \geq 1$

and $\min(\mathrm{R}_{BA}) = 1 \Rightarrow \forall b \in B^{+} \backslash B' : \#(b \bullet \mathrm{R}'_{B'A'}) \geq 1$
and $\forall b \in B'^{+} : \#(b \bullet \mathrm{R}'_{B'A'}) \geq 1$

and $\max(\mathrm{R}_{AB}) = 1 \Rightarrow \forall a \in A^{+} \backslash A' : \#(a \bullet \mathrm{R}'_{A'B'}) \leq 1$
and $\forall a \in A'^{+} : \#(a \bullet \mathrm{R}'_{A'B'}) \leq 1$

and $\max(\mathrm{R}_{BA}) = 1 \Rightarrow \forall b \in B^{+} \backslash B' : \#(b \bullet \mathrm{R}'_{B'A'}) \leq 1$
and $\forall b \in B'^{+} : \#(b \bullet \mathrm{R}'_{B'A'}) \leq 1$

[10] According to the definition of $<_a^{+}$, $[\min(\mathrm{R}_{AB}) = 1 \Rightarrow \min(\mathrm{R}'_{A'B'}) = 1] \wedge [\max(\mathrm{R}_{AB}) = 1 \Rightarrow \max(\mathrm{R}'_{A'B'}) = 1]$ and similarly for R_{BA} and $\mathrm{R}'_{B'A'}$. Additionally $[(p \rightarrow q) \wedge (r \rightarrow s) \wedge (p \rightarrow r)] \rightarrow [p \rightarrow (q \wedge s)]$

$\Rightarrow C' \models I$ and $\forall T \in \mathcal{R}'/\{R', R\}: T \models I$

and $\min(R_{AB}) = 1 \Rightarrow \forall a \in A^+ : \#(a \bullet R'_{A'B'}) \geq 1$

and $\min(R_{BA}) = 1 \Rightarrow \forall b \in B^+ : \#(b \bullet R'_{B'A'}) \geq 1$

and $\max(R_{AB}) = 1 \Rightarrow \forall a \in A^+ : \#(a \bullet R'_{A'B'}) \leq 1$

and $\max(R_{BA}) = 1 \Rightarrow \forall b \in B^+ : \#(b \bullet R'_{B'A'}) \leq 1$

$\Rightarrow C \models I$ and $\forall T \in \mathcal{R}'/\{R', R\}: T \models I$ and $R_{|\mathcal{M}}$[11] $\models I$

$\Rightarrow C \models I$ and $\forall T \in \mathcal{R}/\{R\}: T \models I$ and $R_{|\mathcal{M}} \models I$

$\Rightarrow C \models I$ and $\forall T \in \mathcal{R}: T \models I$

$\Rightarrow \mathcal{M} \models I$

QED

6 Discussion and further research

This paper presented specialization of associations. The first section presented arguments for introducing this new modeling concept. Sections 2 and 3 presented arguments for a formalization of the semantics of this concept. Although specialization of classes is a cornerstone of object-orientation, it has never been applied to associations before. We believe that the reason for this is that associations are only implicitly present in object-oriented programs as client relationships between classes. In a conceptual approach however, associations play a prevalent role in modeling the universe of discourse.

The basic concepts of the object-oriented paradigm have been developed in the context of programming languages. As they were adopted for design and analysis, they were never really re-considered. It seems however a matter of common sense to investigate whether programming concepts can be applied unchanged to design and analysis or whether they need some adaptation to the

[11] $R_{|\mathcal{M}}$ means R as interpreted in model $\mathcal{M}$ as opposed to R as interpreted in $\mathcal{M}'$

raised abstraction level. This paper reconsiders inheritance of inter-object references in particular and adapts it to its conceptual-modeling level counterpart, namely associations. Likewise, in [13, 14] arguments have been given for raising the abstraction level of object interaction modeling techniques for object-oriented conceptual models.

Although the argumentation has been developed for associations between classes in a conceptual model, the same reasoning applies to associations that link a (conceptual-model) class to the data types of its attributes.

The definition of the semantics of association specialization in this paper is based on the interpretation of a conceptual model as a set of constraints on the valid instances of a model. With this interpretation, we have been able to prove that a covariant approach to association inheritance does not lead to inconsistencies: a model that refines associations in a covariant way does not contradict the constraints imposed by the model without refined associations. Further research should however investigate how to resolve the implementation problems raised by covariance.

Another subject of further research is the inheritance of behavior. In [12] a novariant or contravariant approach to inherited state machines has been suggested. However, further research should define how to achieve a covariant approach to behavior inheritance in conceptual modeling.

7 References

1. Booch, Object-oriented Analysis and Design with applications, second edition, Benjamin/Cummings Publishing Company, 1994.
2. Castagna G., Covariance and Contravarianc: conflict without a cause, ACM Transactions on Programming Languages and Systems, Vol. 17, No. 3, May 1995, pp. 431-447

3. Embley D.W., Kurtz B.D., Woodfield S.N. *Object-Oriented Systems Analysis: A Model-Driven Approach*, Yourdon Press, Prentice Hall, Englewood Cliffs, N.J., 1992.
4. Essink L.J.B., Erhart W.J., Object Modeling and System Dynamics in the Conceptualization Stages of Information Systems Development, in F. Van Assche, B. Moulin, C. Rolland (eds.) *Object Oriented Approach in Information Systems*, Proceedings of the IFIP TC8/WG8.1 Working Conference on the Object Oriented Approach in Information Systems, Quebec City, Canada, 28-31 October, 1991, North Holland, 1991, pp. 89-116.
5. Halpin, T.A., Information Modeling and Relational Databases, Morgan Kaufmann Publishers, 2001, 448 pp.
6. Hammer M., McLeod D., Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, Vol. 6, No. 3, September 1981, pp. 351-386.
7. Meyer Bertrand, Object-oriented software construction, Prentice Hall, Englewood Cliffs, N.J., second edition, 1997
8. OMG, The Unified Modeling Language, on-line document via UML resource page, http://www.omg.org/uml/
9. Sernadas C., Resende P., Gouveia P., Sernadas A., In-the-large object-oriented design of Information Systems, in F. Van Assche, B. Moulin, C. Rolland (eds.) *Object Oriented Approach in Information Systems, Proceedings of the IFIP TC8/WG8.1 Working Conference on the Object Oriented Approach in Information Systems*, Quebec City, Canada, 28-31 October, 1991, North Holland, 1991, pp. 209-232
10. Shang David, Is a Cow an Animal? in Object Currents (on-line publication) SIGS New York, Januari 1996 at http://www.visviva.com/transframe/papers/covar.htm
11. Shlear S., Mellor S.J., *Object-Oriented Systems Analysis: Modeling the World in Data*, Prentice Hall, Englewood Cliffs, N.J., 1988.
12. Snoeck M., Dedene G., Generalisation/Specilisation and Role in object-oriented conceptual modeling, *Data and Knowledge Engineering*, 19(2), 1996

13. Snoeck M, Dedene G, Existence dependency: The key to semantic integrity between structural and behavioural aspects of object types, *IEEE Trans. on Software Engineering*, Vol. 24, No. 4, April 1998, pp. 233-251
14. Snoeck M, Poels G., Improving the reuse possibilities of the behavioral aspects of object-oriented domain model. Lecture Notes in Computer Science 1920, in Laendler, A. H. F., S. W. Liddle, and V. C. Storey, ed.: *Conceptual Modeling - ER2000, 19th International Conference on Conceptual Modeling, Salt Lake City,* (Springer Verlag), pp. 423-439 (2000)
15. Szyperski C., Omohundro S., Murer S., Engineering a Programming Language: The type and Class system of Sather, ICSI Techreport TR-93-064, Berkeley California, 1993, available on-line at http://www.icsi.berkeley.edu/~sather/Publications/tr-93-064/
16. Wintraecken, J. J. V. R., Informatie-analyse volgens NIAM in theorie en praktijk, Academic service Den Haag, 1985, 540 pp.