Generic Relationships in Information Modeling

Mohamed Dahchour[1], Alain Pirotte[2], and Esteban Zimányi[3]

[1] Institut National des Postes et Télécommunications,
Av. Allal Al Fassi, Rabat, Morocco
dahchour@inpt.ac.ma
[2] Université catholique de Louvain, IAG School of Management,
1 Place des Doyens, B-1348 Louvain-la-Neuve, Belgium
pirotte@info.ucl.ac.be
[3] Université Libre de Bruxelles, Department of Computer and Network Engineering,
CP 165/15, 50 Av. F. Roosevelt, B-1050 Brussels, Belgium
ezimanyi@ulb.ac.be

Abstract. Generic relationships are abstraction patterns used for structuring information across application domains. They play a central role in information modeling. However, the state of the art of handling generic relationships leaves open a number of problems, like differences in the definition of some generic relationships in various data models and differences in the importance given to some generic relationships, considered as first-class constructs in some models and as special cases of other relationships in other models. To address those problems, we define a list of dimensions to characterize the semantics of generic relationships in a clear and systematic way. The list aims to offer a uniform and comprehensive analysis grid for generic relationships, drawn from a careful analysis of commonalities and differences among the generic relationships discussed in the literature. The usefulness of those dimensions is illustrated by reviewing significant generic relationships, namely, materialization, role, aggregation, grouping, and ownership. Based on those dimensions, a new metamodel for relationships is proposed.

1 Introduction

Information modeling is the activity of creating abstract representations of some aspects of physical and social systems and their environment. Information models are typically built in the early stages of system development, preceding design and implementation. But information models can also be useful even if no system is contemplated: they then serve to clarify ideas about structure and functions in a perception of the world.

Advances in information modeling involve narrowing the gap between concepts in the real world and their representation in information models by identifying powerful abstractions allowing to better represent application semantics (see, e.g., [1,29,31,34,38,46]).

Generic relationships are such powerful abstraction mechanisms. They are high-level templates for relating real-world entities. Well-known generic relationships include the following.

S. Spaccapietra et al. (Eds.): Journal on Data Semantics IV, LNCS 3730, pp. 1–34, 2005.

- *Classification* relates a class with a set of objects sharing the same properties. An object must be an instance of at least one class. It is also known as *is-of*. For example John is an instance of class Person.
- *Association* represents a structural connection among classes. Associations can be *binary* or *n-ary* ($n \geq 3$). An example of a binary association is teaches(Professor,Course). An example of a ternary association is prescription(Doctor,Medicine, Patient).
- *Generalization* relates *superclasses* to their specializations called *subclasses*. Subclasses inherit all properties (attributes, methods, roles, integrity constraints) from their superclasses. Subclasses may define new specific properties. For example, Vehicle is a generalization of Car.
- *Aggregation* associates an *aggregate* (or whole or composite) to its *components* (or parts). It is also known as *part-whole* or *part-of*. For example, Car is an aggregation of Body, Engine, and Wheel.

Generic relationships model patterns abstracting collections of related *specific* relationships. Specific relationships are instances of generic relationships in a particular application. For example, Vehicle◁—Car is a specific generalization with pattern SuperClass◁—SubClass (see Figure 1).

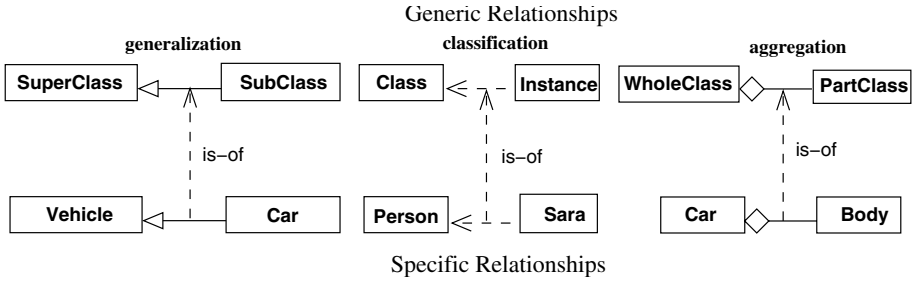


Fig. 1. Generic and specific relationships

Research on information modeling has studied other generic relationships like *materialization* [9], *ownership* [48], *role* [45,10], *grouping* [33], *viewpoint* [32], *generation* [14], *versioning* [3,21], *realization* [25], *transition* [15], and *refinement* [41]. These generic relationships naturally model phenomena typical of complex application domains whose semantics escapes direct representation with classical generic relationships (i.e., association, generalization, classification, and aggregation).

Generic relationships play a central role in information modeling. However, the state of the art of handling them leaves open a number of problems. The semantics of some generic relationships differs among models. Other generic relationships have often been badly understood, underestimated, or merely ignored in some models. In addition, some generic relationships, considered as first-class constructs in some models, are considered as special cases of other relationships in other models. For instance, in UML [41], the aggregation relationship

is considered a special kind of the ordinary association, whereas in most models (e.g., [46,36,19,30,23]) aggregation enjoys a status of its own and comes with more specific features than those defined in UML. Another example concerns the *grouping* relationship relating a collection (e.g., TennisClub) and its members (e.g., TennisPlayer). Grouping was defined in [33] as an independent generic relationship with specific characteristics, while it is just considered as a special case of aggregation, e.g., in [46,19].

Another problem concerns the adequacy of choosing some generic relationships rather than others when modeling an application domain. For example, it can be argued that the relationship between students and employees on the one hand and persons on the other hand is more adequately modeled as a role relationship than as a generalization. Generalization seems more adequate to represent the relationship between males and females on the one hand and persons on the other hand. The idea is that the role relationship captures the temporal and evolutionary aspects of real-world objects (e.g., persons may be students and later become employees), while the usual generalization relationship deals with their more static aspects (e.g., most persons are permanently males or females).

Such difficult questions of adequacy or validity of generic relationships for modeling real-world situations are not directly discussed in this paper, although they are illustrated through a number of examples. Instead, the paper precisely characterizes the structural semantics of generic relationships, to help conceptual database designers precisely evaluate the adequacy of choosing one model rather than another.

We argue that some problems with generic relationships mainly concern the absence of formalizable dimensions or criteria along which the relationships can be characterized in a systematic way. The paper defines such dimensions and illustrates their effectiveness by reviewing some generic relationships. Implementation issues are not presented in this paper. They are discussed in detail in [8].

Notations and Conventions. Table 1 gathers the main notations used in the paper. We use UML [41] notations to specify classes, instances, generalization, instantiation, and aggregation. We add notations to represent concepts that have no equivalent in UML. We prefer to draw associations as boxes with rounded corners rather than using the UML notation. As in UML, instances of relationships are called links.

The rest of the paper is organized as follows. Section 2 presents a preliminary classification of generic relationships. Section 3 presents an overview of our relationship model. Section 4 presents the characteristics of a basic binary relationship (denoted $\mathcal{BBR}$). Section 5 defines a set of dimensions that characterize binary generic relationships. Section 6 reviews several generic relationships in the light of those dimensions. Section 7 gives some guidelines to identify and define new generic relationships. Section 8 presents a new metamodel for generic relationships based on their semantics as presented in Sections 4 and 5. Section 9 summarizes and concludes the paper.

Table 1. Notations and their meanings

Notation	Meaning
$I \dashrightarrow C$ or $I \in C$	I is an instance of class C
C_1—(R)—C_2	R is a binary association between classes C_1 and C_2
S—▷G	S is a subclass of superclass G
W◇—P	whole class W is composed of part class P
A—∗C	abstract class A materializes as concrete class C
$R \rightarrow \circ O$	R is a role class of object class O
$O \prec \cdots P$	owner class O owns property class P
$M \twoheadrightarrow S$	M is a member class of set S
$\mathcal{BBR}$	the basic binary relationship
$C_1 \overset{R}{\text{——}} C_2$	R is a binary relationship between classes C_1 and C_2
$\rho_C(R)$	the role played by class C in relationship R
$C_1(v_1, v_2) \overset{R}{\text{——}} C_2$	cardinality of role $\rho_{C_1}(R)$ is (v_1, v_2)
$\pi_C(R)$	the set of instances of C participating in R with role $\rho_C(R)$
$R(C_1, C_2)$	R relates C_1 and C_2
$R(c_1, c_2)$	there is an instance of R (a link) relating c_1 and c_2
$R_1 \otimes R_2$	relationships R_1 and R_2 are exclusive
$\rho_C(R_1) \otimes \rho_C(R_2)$	roles $\rho_C(R_1)$ and $\rho_C(R_2)$ are exclusive
$R_1 \subseteq R_2$	relationship R_1 is inclusive in relationship R_2
$\rho_C(R_1) \subseteq \rho_C(R_2)$	role $\rho_C(R_1)$ is inclusive in role $\rho_C(R_2)$
$C_0 \overset{R \mid d}{\text{——}} \{C_1, \ldots, C_n\}$	partition of R in classes C_i according to discriminator d

2 Classification of Generic Relationships

Generic relationships can be classified along the following three orthogonal dimensions, as depicted in Figure 2: (1) degree, (2) structurality and dynamicity, and (3) dependency on application domains.

Degree. It is the number of participating classes in a relationship. A relationship of degree two is said to be *binary*, and one of degree n ($n \geq 3$) is *n-ary*. Examples of binary generic relationships include:

- *classification* (of pattern Class◂- -Instance) relates an instance (e.g., Sarah) to its class (e.g., person);
- *generalization* (of pattern SuperClass◁—SubClass) relates a superclass (e.g., persons) to its subclasses (e.g., males and females).
- *materialization* (of pattern Abstract—∗Concrete) [39,9] relates a class of categories (e.g., models of cars) with a class of more concrete objects (e.g., individual cars);
- *ownership* (of pattern Property· · ·≻Owner) [48] relates an owner class (e.g., persons) and a property (e.g., cars) possessed by their objects;
- *aggregation* (of pattern WholeClass◇—PartClass) [30,46] relates composites (e.g., cars) to their components (e.g., body and engine);

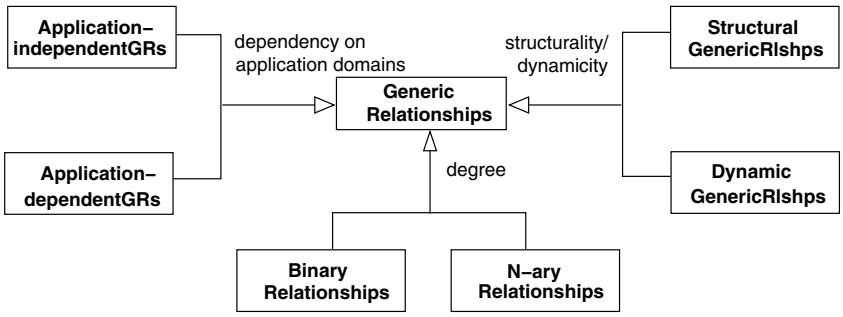


Fig. 2. Classification of generic relationships

- *role* (of pattern ObjectClass$\circ\!\leftarrow$RoleClass) [10] relates an object class (e.g., persons) and a role class (e.g., employees), describing dynamic states for the object class;
- *grouping* (of pattern MemberClass$\rightarrow\!\!\!\!\rightarrow$SetClass) [33] relates a member class (e.g., players) and a grouping class (e.g., teams);
- *viewpoint* (of pattern Class$—\!\!\odot$View) [32] represents partial information about a class viewed from a particular standpoint.

Although most generic relationships are binary, there are situations that are best modeled with n-ary generic relationships. Examples of n-ary generic relationships include the following:

- *view*(Actor,Concept,Facet) [26] relating an actor who sees a concept in a particular facet. Unlike the *viewpoint* relationship above which is closer to the view concept in databases, relationship *view* was defined in the context of requirements engineering;
- *dependency*(Depender,Dependee,Dependum) [49] relating an agent (depender) depending on the other (dependee) for something (dependum) in order that the former may attain some goals.

Structural and dynamic generic relationships. Structural generic relationships are relationships whose semantics can be expressed in terms of static constraints and rules. They are also referred to as *organizational* relationships. Examples of structural generic relationships include the usual classification (where an instance cannot change its class), generalization, aggregation, materialization, grouping, and ownership.

By contrast, dynamic relationships involve some dynamic behavior. Examples include: *dynamic classification* [11], where an instance can dynamically change its class, *generation* [14] where an instance (or a set of instances) of the input class produces an instance (or a set of instances) of the output class; *versioning* [3], which relates an object class and its time-varying versions, modeling various states of the object class; *realization* [25], a variant of classification that allows an object to add structure to that defined by its class; *transition* [15],

which keeps track of migrations of instances from source classes to target classes; refinement [41], which specifies that a class is at a finer degree of abstraction than another class; and *role* [45,10], relating an object class with a role class describing its dynamic states.

Dependency on application domains. Generic relationships can be *application independent* or *application specific*. Examples of generic relationships with application-independent semantics include generalization, aggregation, classification, materialization, grouping, role, versioning, and view. An example of application-specific generic relationships is *BinaryDirectedLink* that relates two nodes (or a node and a link) of an hypermedia graph of a document [44]. Another example deals with architectural design [38] where, for example, a relationship *hasGeometry* relates a symbolic object and a geometric object; *adjacentTo* relates two objects unrelated by aggregation and whose distance is less than a specific threshold; *connectedTo* is similar to *adjacentTo*, but the related objects have overlapping volumes. In business and organizational applications, a network of dependency relationships (e.g., *goal dependency*, *task dependency*, *resource dependency*, and *softgoal dependency*) among actors was proposed in [49]. In the domain of resource management, relationship *production*(Resource,Producer,Consumer) can be defined among resources, their producers, and their consumers.

This paper focuses on structural generic relationships with application-independent semantics. The term *generic relationships* will henceforth denote those structural generic relationships.

3 Our Relationship Metamodel: Overview

Our relationship metamodel, depicted in a simplified way in Figure 3, encompasses three kinds of generic relationships: (i) the basic binary relationship $\mathcal{BBR}$, (ii) binary generic relationships like generalization, aggregation, materialization, and association, and (iii) n-ary generic relationships. This section gives an overview of each of those relationships. Their characteristics are presented in detail in Sections 4 and 5. A more elaborate version of the metamodel in Figure 3 is given in Section 8.

The basic binary relationship $\mathcal{BBR}$. It is a generic relationship (of pattern MetaClass——MetaClass) between two metaclasses denoted by MetaClass. MetaClass is the generic metaclass for all application classes. $\mathcal{BBR}$ defines a basic semantics for relationships comprising cardinalities, existence dependency, symmetry, instance transitivity, exclusiveness, inclusion, and attribute propagation. $\mathcal{BBR}$ can be directly instantiated as specific relationships between application classes, without going through generic relationships like generalization, aggregation, or materialization. These specific relationships, like Employee—works—Company, are called *associations* in UML. We will henceforth use that terminology for those relationships.

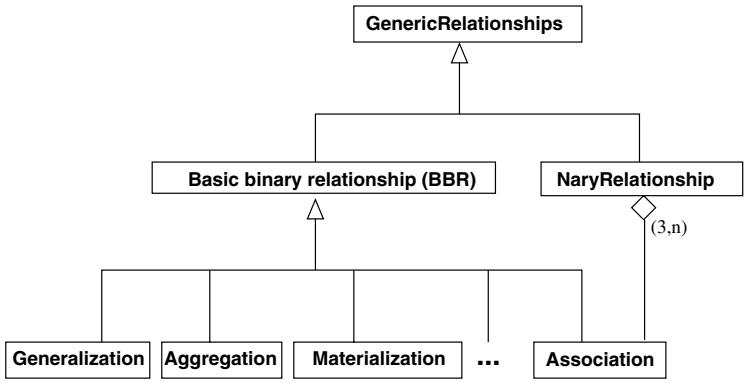


Fig. 3. A simplified view of our metamodel for relationships

Binary generic relationships. Generic relationships like generalization, aggregation, and materialization need the basic semantics of $\mathcal{BBR}$ in addition to their specific semantics. It is thus suitable that generic relationships inherit the semantics of $\mathcal{BBR}$ by subclassing. Hence, the role of $\mathcal{BBR}$ is twofold: directly represent associations and factor out the common semantics of all binary generic relationships.

Generic relationships may refine or redefine the inherited semantics to fit their specific semantics. For instance, cardinality can be inherited from $\mathcal{BBR}$, and refined by generalization as follows: SubClass (1,1)—▷(0,1) SuperClass. Thus, in all specific generalizations (e.g., Car—▷Vehicle), an instance of the subclass corresponds to exactly one instance of the superclass and an instance of the superclass corresponds to at most one instance of the subclass.

N-ary generic relationships. An n-ary relationship relates n classes, for $n \geq 3$. N-ary relationships are more complex than binary ones, they have been much less studied, and have often been poorly understood in practice. Several data models (e.g., OML [12], ORM [18]) do not directly support n-ary relationships. Other models advise against the usage of n-ary relationships, like in the reference manual of UML 2.0 [41], page 472: "In general it is best to avoid n-ary associations, because binary associations are simpler to implement and they permit navigation". It is often argued that n-ary relationships are not frequent in real-world applications, but this point of view is rather a testimony of the typical sophistication of the models produced by current practice in conceptual database design. In practice, n-ary relationships are often represented, and often approximately, by some of their binary projections to be supplemented by consistency constraints. The only correct general way to do without an n-ary relationship is to reify it as a new class with n binary relationships to the given n classes. Our relationship metamodel follows this approach.

We define the semantics of n-ary relationships along a subset of the basic semantics of $\mathcal{BBR}$ (see Section 4). Namely, this semantics includes cardinality,

exclusiveness, inclusion, existence dependency, and attribute propagation. The semantics of an n-ary relationship can be defined in terms of the n binary relationships representing it coupled with some dependency constraints. In Figure 3, n-ary relationships are represented by the metatype NaryRelationship that is defined as an aggregate of three or more binary associations represented by metatype Association. Details concerning the decomposition of n-ary relationships into binary equivalent structures can be found in, e.g., [28,43,20].

4 The Basic Binary Relationship

We define the semantics of the basic binary relationship $\mathcal{BBR}$ along several dimensions, including cardinality, existence dependency, symmetry, instance transitivity, exclusiveness, inclusion, and attribute propagation mechanisms. These characteristics are defined in detail in Sections 4.1 to 4.7. They are independent of one another except for some consistency constraints to be defined for combining exclusiveness and inclusion. The completeness of our set of dimensions, although intuitively very important, cannot be proved. It can only be addressed in a pragmatic and empirical manner. The list of characteristics presented in this paper aims to offer a uniform and comprehensive analysis grid for generic relationships; it was drawn after carefully analyzing commonalities and differences among the generic relationships discussed in the literature.

4.1 Cardinality

The cardinality dimension constrains the number of relationship links in which an object can participate. Let R be a relationship associating classes C_1 and C_2. A cardinality (min, max) at the side of C_1 means that each instance of C_1 must participate in at least min and at most max links of R at all times. The most frequent cardinalities are: (0,1) (i.e., at most one), (1,1) (i.e., exactly one), (0,n) (i.e., any number, the unconstrained case), and (1,n) (i.e., at least one). We find this traditional definition more intuitive and expressive than that of UML.

4.2 Existence Dependency

Existence dependency characterizes whether or not an object can exist independently of related objects. There are two typical cases:

- *dependency*, meaning that the existence of an object of C_1 depends on the existence of related objects of C_2. This is known as *mandatory* participation in ER modeling. It is expressed by a minimum cardinality of 1 at the side of C_1;
- *independency*, meaning that the existence of an object of C_1 is independent of the existence of related objects of C_2. This is known as *optional* participation in ER modeling. It is expressed by a minimum cardinality of 0 at the side of C_1.

Existence dependency also specifies how insertion or deletion of one object may influence the existence of connected objects. Let o_1 and o_2 be two objects related by link r. With respect to the deletion operations, there are three main options to maintain the existence dependency:

- *default deletion:* the deletion of an object implies the deletion of all its links (e.g., the deletion of o_1 implies the deletion of r);
- *cascade deletion:* the deletion of an object implies the deletion of all its links as well as all other objects involved in those links (e.g., the deletion of o_1 implies the deletion of r and o_2);
- *restrict deletion:* the deletion of an object is prohibited if the object is involved in at least one link (e.g., the deletion of o_1 is disallowed while link r exists).

Those deletion options can be associated to each role of a given relationship. For example, in Figure 4, *cascade* deletion associated with role employs means that the deletion of a department implies the deletion of all its employees. The *default* deletion associated with role worksOn states that the deletion of an employee only implies the deletion of its link to the related department.

Fig. 4. The cascade and default deletion options

Deletion options must be carefully chosen to avoid inconsistencies. For the example in Figure 4, assume now that the deletion option on role worksOn is *restrict* instead of *default*. When a department is deleted, option *cascade* on role employs requires the deletion of all related employees, while option *restrict* on role worksOn states that those employees cannot be deleted. Thus a contradiction arises.

Existence dependency is sometimes referred to as *referential integrity*. In object models, this means that for any object o_1 containing a reference to an object o_2, the referred object o_2 must indeed exist. In systems where referential integrity is not automatically ensured, the problem of *dangling pointers* may arise if a referred object is deleted.

In the relational model, referential integrity is an inclusion constraint between a set of attributes (called *foreign key*) of a *child* relation and the attributes forming the primary key of a *parent* relation. For the child relation, this concerns insert and update operations. Various repair actions can be specified to avoid violations resulting from deletions and updates in the parent relation:

- *cascade:* in case of update, the new values in the key are propagated to the referencing children, whereas in case of deletion the referencing children are also deleted;

- *set null:* the foreign key attributes in the referencing tuples of the child relation are set to null;
- *set default:* the foreign key attributes in the referencing tuples of the child relation are set to a given default value;
- *no action:* referential integrity remains violated and, if no other operation is executed to correct the mismatch of the corresponding tuples, the complete transaction is rolled back.

Some relational database systems introduce another referential action called *restrict*. Its semantics forbids any change (update or delete) to the primary key of a parent tuple as long as there are referencing child tuples.

4.3 Symmetry

A binary relationship R associating classes C_1 and C_2 is *symmetric* iff $\forall c_1 \in C_1 \, \forall c_2 \in C_2 \; (R(c_1, c_2) \Leftrightarrow R(c_2, c_1))$. Relationships that are not symmetric are called *asymmetric*. Most binary relationships are asymmetric.

The properties of symmetry and asymmetry are particularly relevant for recursive relationships (i.e., where C_1 and C_2 are the same class). Examples of symmetric recursive relationships include siblingOf(Person,Person) and jointlyTaxedWith(Person,Person). Symmetric recursive relationships are also said to be *reflexive*. Examples of asymmetric recursive relationships include supervises(Employee,Employee), assembly(Part,Part), and ancestorOf(Person,Person). Asymmetric recursive relationships are also said to be *irreflexive*.

4.4 Instance Transitivity

Let R be a binary recursive relationship whose instances relate two instances of class C. R is *instance transitive* iff $\forall c_1, c_2, c_3 \in C \; (R(c_1, c_2) \wedge R(c_2, c_3) \Rightarrow R(c_1, c_3))$. For instance, relationship ancestorOf(Person,Person) is instance transitive. Instance transitivity is different from class transitivity, which is presented in Section 5.3.

4.5 Exclusiveness

The *exclusiveness* dimension for binary relationships can be defined for both relationships and roles [18]. As for notation, exclusiveness is represented by a dashed line labeled with the symbol $\otimes$.

Relationship exclusiveness. Exclusiveness between two relationships R_1 and R_2 is defined for relationships relating the same classes C_1 and C_2. It means that the set of links of both relationships are disjoint.

For example, Figure 5 shows two exclusive relationships borrows and reserves: a student cannot simultaneously borrow and reserve the same book (i.e., borrows $\cap$ reserves $= \emptyset$).

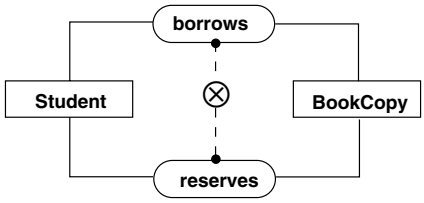


Fig. 5. Exclusiveness between relationships borrows and reserves

Role exclusiveness. Let R_1 and R_2 be two relationships sharing the same class C_0 at one end. Let $\rho_{C_0}(R_i)$ denote the role played by C_0 in R_i and $\pi_{C_0}(R_i)$ denote the set of instances of C_0 participating in R_i. Exclusiveness between roles $\rho_{C_0}(R_1)$ and $\rho_{C_0}(R_2)$ means that $\pi_{C_0}(R_1)$ and $\pi_{C_0}(R_2)$ are disjoint. Formally, $\rho_{C_0}(R_1) \otimes \rho_{C_0}(R_2) \Leftrightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset$.

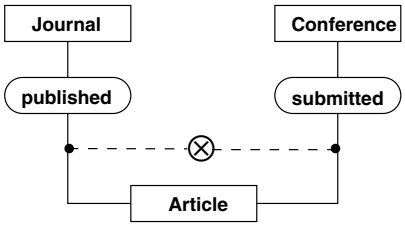


Fig. 6. Exclusiveness between roles $\rho_{\mathsf{Article}}(\mathsf{published})$ and $\rho_{\mathsf{Article}}(\mathsf{submitted})$

For example, the exclusiveness between roles in Figure 6 means that an article cannot simultaneously be submitted to a conference and be published in a journal. In this example, the end classes Journal and Conference of relationships published and submitted, respectively, are distinct. An example of exclusiveness between roles involved in relationships sharing the same end classes is shown in Figure 7: the exclusiveness between roles $\rho_{\mathsf{Apartment}}(\mathsf{rent})$ and $\rho_{\mathsf{Apartment}}(\mathsf{sell})$ means that an apartment cannot simultaneously be rented and sold to clients.

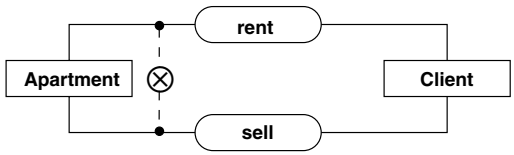


Fig. 7. Exclusiveness between roles $\rho_{\mathsf{Apartment}}(\mathsf{rent})$ and $\rho_{\mathsf{Apartment}}(\mathsf{sell})$

4.6 Inclusion

Like exclusiveness, the *inclusion* dimension for binary relationships can be defined for both relationships and roles. Graphically, inclusion of relationship R_1 in R_2 is represented by a dashed arrow labeled $\subseteq$ with its origin in R_1 and its destination in R_2.

Relationship inclusion. Inclusion of relationship R_1 in relationship R_2 is defined for relationships sharing the same classes C_1 and C_2 at their ends. It means that the set of links of R_1 is a subset of the set of links of R_2.

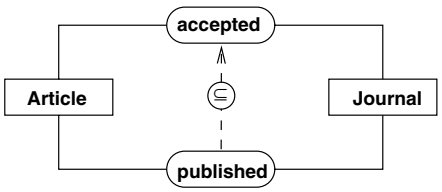


Fig. 8. Inclusion of relationship published in relationship accepted

For example, Figure 8 shows inclusion of relationship published in relationship accepted. This means that an article published in a journal has necessarily been accepted for publication in that journal (i.e., published $\subseteq$ accepted).

Role inclusion. Inclusion of role $\rho_{C_0}(R_1)$ in role $\rho_{C_0}(R_2)$ is defined for relationships sharing the same class C_0 at one end. It means that the set of instances of C_0 participating in role R_1 is a subset of the set of instances of C_0 participating in role R_2. Formally, $\rho_{C_0}(R_1) \subseteq \rho_{C_0}(R_2) \Rightarrow \pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2)$.

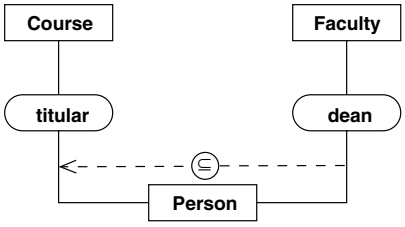


Fig. 9. Inclusion of role $\rho_{\mathsf{Person}}(\mathsf{dean})$ in role $\rho_{\mathsf{Person}}(\mathsf{titular})$

For example, Figure 9 shows inclusion of role $\rho_{\mathsf{Person}}(\mathsf{dean})$ in role $\rho_{\mathsf{Person}}(\mathsf{titular})$. This means that a faculty dean must be titular of a course. In this example, the end classes Faculty and Course of relationships dean and titular, respectively, are distinct. An example of inclusion between roles participating in relationships sharing the same end classes is shown in Figure 10.

Role $\rho_{\mathsf{Student}}(\mathsf{practices})$ is included in role $\rho_{\mathsf{Student}}(\mathsf{registers})$, meaning that a person practicing a sport has necessarily registered in a sport. That allows John to practice Tennis while being registered in Football. If it is required that each person who practices a sport should necessarily register for that sport, then the constraint should be an inclusion of relationship practices in relationship registers.

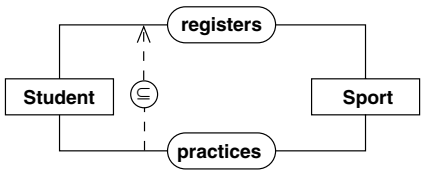


Fig. 10. Inclusion of role $\rho_{\mathsf{Student}}(\mathsf{practices})$ in role $\rho_{\mathsf{Student}}(\mathsf{registers})$

Consistency rules. The following additional rules hold for exclusiveness and inclusion constraints on relationships R_1 and R_2:

- relationship inclusion and relationship exclusiveness cannot coexist, because $R_1 \subseteq R_2 \Rightarrow R_1 \cap R_2 \neq \emptyset$ and conversely $R_1 \cap R_2 = \emptyset \Rightarrow (R_1 \not\subseteq R_2) \wedge (R_2 \not\subseteq R_1)$;
- relationship inclusion and role exclusiveness cannot coexist, because $R_1 \subseteq R_2 \Rightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) \neq \emptyset$ and conversely $\pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset \Rightarrow (R_1 \not\subseteq R_2) \wedge (R_2 \not\subseteq R_1)$;
- role inclusion and role exclusiveness cannot coexist, because $\pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2) \Rightarrow \pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) \neq \emptyset$ and conversely $\pi_{C_0}(R_1) \cap \pi_{C_0}(R_2) = \emptyset \Rightarrow (\pi_{C_0}(R_1) \not\subseteq \pi_{C_0}(R_2)) \wedge (\pi_{C_0}(R_2) \not\subseteq \pi_{C_0}(R_1))$.
- however, role inclusion and relationship exclusiveness may coexist, because $\pi_{C_0}(R_1) \subseteq \pi_{C_0}(R_2) \not\Rightarrow R_1 \cap R_2 \neq \emptyset$ and conversely $R_1 \cap R_2 = \emptyset \not\Rightarrow \pi_{C_0}(R_1) \not\subseteq \pi_{C_0}(R_2)$.

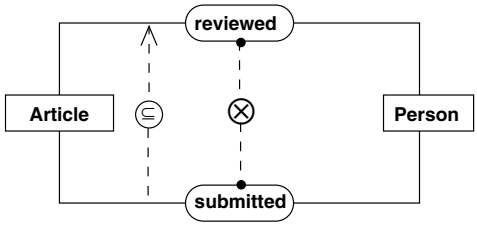


Fig. 11. Coexistence between role inclusion and relationship exclusiveness

For example, in Figure 11 role $\rho_{\mathsf{Article}}(\mathsf{submitted})$ is included in role $\rho_{\mathsf{Article}}(\mathsf{reviewed})$ (a reviewed paper has necessarily been submitted), while relationships submitted and reviewed are mutually exclusive (an article cannot be submitted and reviewed by the same person).

4.7 Propagation Mechanisms

Given a binary relationship R associating classes C_1 and C_2, attributes can propagate from C_1 to C_2 and vice versa. We define two kinds of propagated attributes: *derived* attributes and *computed* attributes.

An attribute of class C_1 is said to be derived from C_2 through R if it is inherited as is from C_2 to C_1. An attribute a_1 of class C_1 is said to be computed from attributes $a_{2_1}, \ldots, a_{2_p}$ $(p \geq 1)$ of C_2 through R if the value of a_1 for a given object c_1 of C_1 is computed from values of attributes $a_{2_1}, \ldots, a_{2_p}$ of objects belonging to C_2 that are related to c_1 via R. This means that value(a_1)= f(value(a_{2_1}), ..., value(a_{2_p})). Function f is an aggregate operator such as +, -, *, min, max, avg.

For example, attribute publicationDate of class Article can be derived from attribute issueDate of class Journal via relationship published, since its value for each article is the same as its value for the issue of the journal in which the article is published. Similarly, the number of pages of a journal can be computed as the sum of the number of pages of its articles. Also, the average grade of a class can be computed as the average of individual grades of all students of that class.

Propagation mechanisms are expressed in ODMG [5] by means of *path expressions*. For example, the publication date of an article a1 can be expressed by the path expression a1.j1.issueDate that returns the publication date of journal j1 where a1 appeared. Such propagations are not specific to relationships. Derived and computed attributes can also be defined within a single class. An usual example is attribute age of class Person that can be derived from attribute birthDate of the same class.

5 Binary Generic Relationships

This section describes the characteristics of binary generic relationships in a systematic way. Part of these characteristics are inherited from the basic relationship $\mathcal{BBR}$ by subclassing. As mentioned earlier, some of these properties, like cardinality, can be redefined by some generic relationships to conform with their particular semantics. Another part of the characteristics of generic relationships is described along several dimensions like class- and instance-level semantics, composition, class transitivity, class nonrecursivity, multiplicity, and partitioning. These dimensions are described in detail in Sections 5.1 to 5.9.

Although these dimensions have been identified by carefully studying a substantial collection of generic relationships, we cannot claim that they are exhaustive. The list remains open to other dimensions that could be identified by exploring other generic relationships.

The following notations and assumptions are needed to formally define some dimensions below:

- Let $R(MC_1, MC_2)$ be a predicate stating that R is a binary generic relationship between two metaclasses MC_1 and MC_2. Specific relationships, which

are instances of R, have the same name and are defined between classes denoted by C_1 and C_2. Thus, $R(C_1, C_2)$ means that there is a specific relationship R between classes C_1 and C_2 where C_1 is an instance of MC_1 and C_2 is an instance of MC_2.
- The role played by MC_1 in R, noted $\rho_{MC_1}(R)$, is the same as the role played by C_1 in R, noted $\rho_{C_1}(R)$. For example, the whole role played by WholeClass in WholeClass◇—PartClass is the same as the role played by Car in the specific aggregation Car◇—Body.
- $C \in MC$ means that class C is an instance of metaclass MC.

5.1 Class- and Instance-Level Semantics

The semantics of generic relationships concerns both classes and instances of these classes. Consequently, comprehensive semantics must deal with both the *class level* and the *instance level* in a coordinated manner.

For example, the class-level semantics of generalization states that:

- a class can have several superclasses and several subclasses;
- each class inherits all properties from its superclasses;
- conflicts induced by multiple inheritance are avoided with a specified strategy;
- each class has a (1,1) cardinality regarding each of its superclasses and a (0,1) cardinality regarding each of its subclasses.

At the instance level, the generalization relationship expresses the following semantics:

- an instance of a class C cannot be an instance of another class that is not direct or indirect superclass of C[1];
- an instance cannot have additional properties than those of its class[2].

5.2 Composition

Generic relationships can be involved in compositions, where a class plays several roles of the same generic relationship R in several specific relationships based on R, as schematized in Figure 12(a). Formally, a generic relationship R can be composed iff $\exists C_1, C_2, C_3$ such that $R(C_1, C_2) \wedge R(C_2, C_3)$ where $C_1 \in MC_1, C_2 \in MC_2, C_2 \in MC_1$, and $C_3 \in MC_2$. C_2 plays at the same time role $\rho_{MC_1}(R)$ and role $\rho_{MC_2}(R)$.

An example of composition of generalizations is Person◁—Student◁—GraduateStudent, where Student is at the same time a superclass of GraduateStudent and a subclass of Person (see Figure 12(b)). Similarly, in the composition of aggregations Car◇—Body◇—Door, Body is at the same time a composite of Door and a component of Car.

[1] This is possible in models allowing multiple classification, like Telos [35] or MADS [37], where an object can be an instance of several classes not related, directly or indirectly, by the generalization link.

[2] This restriction is overcome with the realization relationship [25].

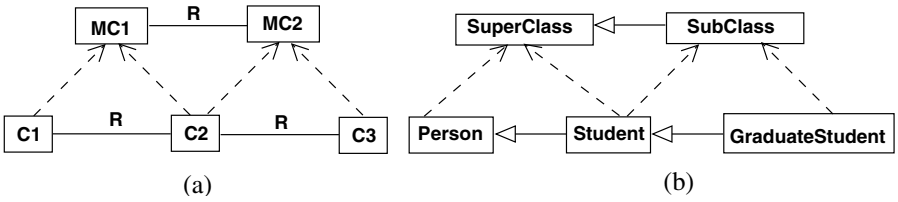


Fig. 12. Composition of relationships

5.3 Class Transitivity

Some generic relationships that can be composed may be class transitive. A generic relationship R is class transitive iff $\forall C_1, C_2, C_3\ R(C_1, C_2) \wedge R(C_2, C_3) \Rightarrow R(C_1, C_3)$. For example, generalization is class transitive: the generalizations Person◁—Student◁—GraduateStudent imply Person◁— GraduateStudent. By contrast, aggregation is not class transitive in general.

5.4 Class Nonrecursivity

A generic relationship R is said to be *class nonrecursive* if R does not hold at the class level between two occurrences of the same class. For example, generalization is class nonrecursive, since a class cannot be a subclass of itself, while aggregation may be class recursive (e.g., Program◇—Program).

5.5 Multiplicity

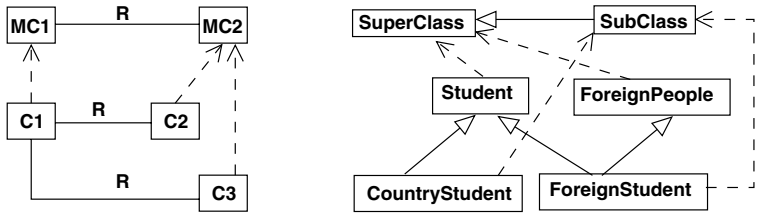
A role in a generic relationship is said to be *multiple*[3] if the same class can participate with that role in several instances of the generic relationship (see Figure 13(a)). Formally, role $\rho_{MC_1}(R)$ is multiple iff $\exists C_1, C_2, C_3$ such that $R(C_1, C_2) \wedge R(C_1, C_3)$ where $C_1 \in MC_1, C_2 \in MC_2$, and $C_3 \in MC_2$. Multiplicity of role $\rho_{MC_2}(R)$ can be defined in a similar way. Figure 13(a) shows an example of multiplicity for role $\rho_{MC_1}(R)$.

Most generic relationships allow multiplicity in each role. For example, with generalization, a class can have several superclasses and several subclasses as shown in Figure 13(b). Also, with aggregation, a composite can have several components and a component can be part of several composites.

5.6 Partitioning

Generalization and aggregation were defined so far as binary relationships. However, it is often natural to group several binary relationships that involve the same superclass for generalizations, or the same component or the same composite for aggregations. Such groupings add semantics to the semantics of the

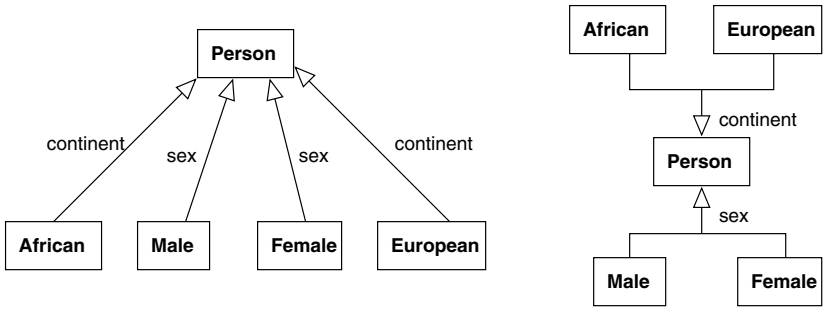
[3] This definition of multiplicity should not be confused with its use, e.g., in UML, as a synonym of *cardinality*.



(a) (b)

Fig. 13. Multiplicity of relationships

participating binary relationships. As in UML, such groupings are referred to as *partitions*[4]. Each partition, together with the binary generalizations and aggregations taking part in it, carry a unique discriminator (or label). A partition for a generic relationship R is noted $C_0 \overset{R\,|\,d}{\text{———}} \{C_1, \ldots, C_n\}$ where the $C_0 \overset{R}{\text{———}} C_i$ are specific binary relationships that are instances of R, and d is the discriminator of the partition.



(a) (b)

Fig. 14. Partitions for generalizations

Figure 14(b) shows two partitions for generalizations, Person$\overset{\text{sex}}{\triangleleft\!\text{———}}${Male, Female} and Person$\overset{\text{continent}}{\triangleleft\!\text{———}}${African, European}, obtained by grouping binary generalizations in Figure 14(a), with discriminators sex and continent, respectively. Similarly, Figure 15(b) shows two partitions obtained by grouping binary aggregations with discriminators space and time.

Partitions of generalizations can be characterized along the usual dimensions of *totality* and *exclusiveness*, resulting in four combinations: (total, exclusive),

[4] The term *partition* has a specific meaning in set theory. In our context, the term *grouping* could be more appropriate, but we avoid it because it denotes another generic relationship.

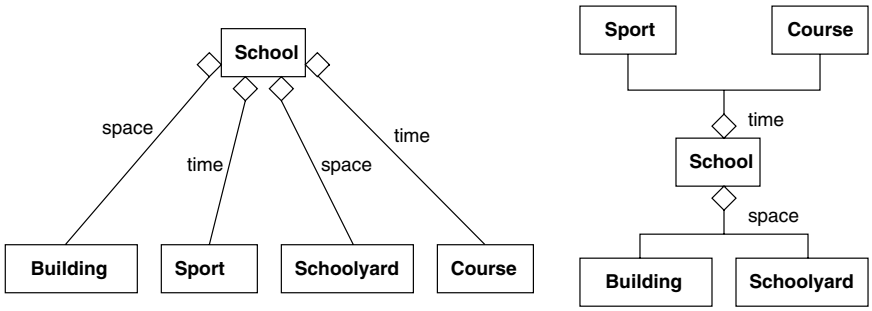


Fig. 15. Partitions for aggregations

(partial, exclusive), (total, overlapping), and (partial, overlapping). More formally, let $\mathcal{P} = C_0 \overset{d}{\triangleleft\!\!-} \{C_1, \ldots, C_n\}$ be a partition of a superclass C_0 into a set of subclasses $C_1, \ldots, C_n$ according to discriminator d. The dimensions above are defined as follows.

- *Total:* every instance of the superclass is an instance of at least one subclass in $\mathcal{P}$. Formally, $\mathcal{P}$ is total iff $\forall c \in C_0\ \exists i \in [1,n]\ (c \in C_i)$.
- *Partial:* an instance of the superclass needs not be an instance of a subclass within $\mathcal{P}$. Formally, $\mathcal{P}$ is partial if it may be the case that $\exists c \in C_0\ \forall i \in [1,n]\ (c \notin C_i)$.
- *Exclusive:* an instance of the superclass may be an instance of no more than one subclass within $\mathcal{P}$. Formally, $\mathcal{P}$ is exclusive iff $\forall i \in [1,n]\ \forall c \in C_i\ \nexists j \in [1,n]\ (i \neq j \wedge c \in C_j)$.
- *Overlapping:* an instance of one subclass may simultaneously be an instance of another subclass in $\mathcal{P}$. Formally, $\mathcal{P}$ is overlapping if it may be the case that $\exists c \in C_0\ \exists i,j \in [1,n]\ (i \neq j \wedge c \in C_i \wedge c \in C_j)$.

Some authors (e.g., [45]) distinguish *static* from *dynamic* partitions. Roughly, static partitions correspond to total and exclusive partitions, and dynamic partitions correspond to total and overlapping partitions.

5.7 Exclusiveness

This characteristic is inherited from relationship $\mathcal{BBR}$ which defines two categories of exclusiveness: relationship exclusiveness and role exclusiveness. However, generic relationships with the multiplicity property for a role may only involve exclusiveness between roles.

For example, in the aggregations Proceedings$\diamond$—Article—$\diamond$Journal, Proceedings$\diamond$—Article and Journal$\diamond$—Article can be exclusive, with the meaning that the same article cannot appear both in conference proceedings and in a journal. Similarly, in the materializations Video$*$—Movie—$*$ DVD, Movie—$*$DVD

and Movie—∗Video can be considered as exclusive if a movie can materialize either in a DVD or in a Video but not in both at the same time.

5.8 Inclusion

Like exclusiveness, this characteristic is inherited from relationship $\mathcal{BBR}$. Generic relationships with the multiplicity property for a role may only involve inclusion between roles.

For example, in the generalizations PhDStudent—▷Person◁—TeachingAssistant, generalization Person◁—TeachingAssistant is inclusive in Person ◁—PhDStudent if a teaching assistant is necessarily a PhD student. Similarly, in the materializations Video∗—Movie—∗DVD, materialization Movie —∗DVD is inclusive in Movie—∗Video if a movie can materialize as a DVD only if it is already materialized as a Video. In the groupings PoliticalParty↞—Deputy↠Parliament, Deputy↠Parliament is inclusive in Deputy↠PoliticalParty if a deputy in a parliament necessarily belongs to a political party.

5.9 Propagation

Most generic relationships allow propagating structure and behavior from one participant to another. This is carried out by *inheritance* or by *delegation*. In some cases, propagation is unidirectional. For example, in generalization, subclasses (totally or partially) inherit attributes and methods from superclasses. In aggregation, composites can access some properties of their parts (and vice versa) by delegation. For example, Car inherits the color attribute of its component Body.

Inheritance by delegation [27] can be characterized as follows. Assume a message m(arguments) is sent to an object o_1 and method m is not defined in o_1's class. According to the usual message-passing semantics, the message handler would signal an error. But, if the object o_1 was related to an object o_2 with respect to some semantic relationship, it could make sense to find out whether o_2 would be able to return a semantically meaningful response by executing method m. Thus, the object o_1 can be seen as delegating the execution of method m to its related object o_2.

6 A Review of Some Generic Relationships

This section reviews some generic relationships and defines their semantics in the light of the dimensions defined in Section 5. Of course, the goal here is not to describe in detail that semantics, but rather to illustrate the effectiveness of our dimensions.

6.1 Materialization

Materialization [9] is a binary relationship with pattern Abstract—∗ Concrete relating a class of abstract objects and a class of more concrete objects, where each abstract object can be viewed as a category characterizing a subset of the concrete objects.

In the example of Figure 16, CarModel is the abstract class of materialization CarModel—∗Car and Car is its concrete class. CarModel represents information typically displayed in the catalog of car dealers, while class Car represents information about individual cars. Figure 17 shows an instance FiatRetro of CarModel and an instance Nico's Fiat of Car, of model FiatRetro.

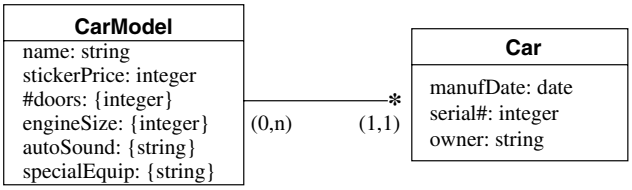


Fig. 16. An example of materialization

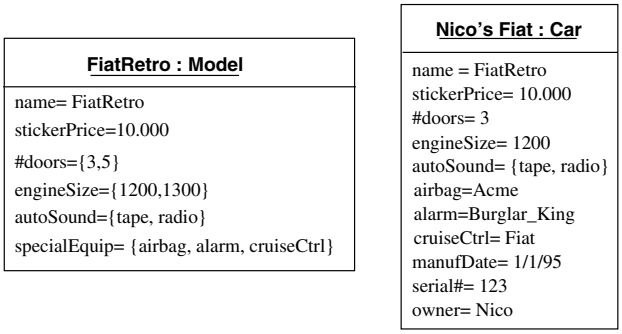


Fig. 17. Instances of CarModel and Car classes from Figure 16

Cardinality. Intuitively, the materialization CarModel—∗Car means that every concrete car (e.g., Nico's Fiat) has exactly one model (e.g., Fiat-Retro), while there can be any number of cars of a given model. Most real-world examples of materialization have cardinality (1,1) at the side of the concrete class and cardinality (0,n) at the side of the abstract class, although the latter cardinality can be further constrained.

Dependency. In a materialization, the deletion of an abstract instance induces the deletion of its associated concrete instances. In the materialization of Figure 16, when model FiatRetro is deleted, all instances of Car associated to that

model, e.g., Nico's Fiat, are also deleted. On the other hand, the deletion of a concrete instance induces the deletion of its associated abstract instance only if the minimal cardinality at the abstract side of the materialization is 1. For example, if in CarModel—∗Car the minimal cardinality of CarModel is 1, meaning that a car model has associated at least one car, then the deletion of the last car of a model implies the deletion of that car model.

Attribute propagation. Some information in a concrete instance is naturally viewed as obtained from its associated abstract instance. For example, in Figure 17, Nico's Fiat directly inherits the name and stickerPrice of its model FiatRetro. Further, Nico's Fiat has attributes #doors, engineSize, and autoSound whose values are selections among the options offered by multivalued attributes with the same name in FiatRetro. For example, the value of engineSize for Nico's Fiat is taken from the possible values of the engineSize in FiatRetro. Finally, the value {airbag, alarm, cruiseCtrl} of attribute specialEquip for FiatRetro means that each car of model FiatRetro comes with three pieces of special equipment: an air bag, an alarm system, and a cruise-control system. Thus, Nico's Fiat has three new attributes named airbag, alarm, and cruiseCtrl, whose suppliers are, respectively, Acme, Burglar_King, and Fiat. Other FiatRetro cars may have different suppliers for their special equipment and cars of models other than FiatRetro may have a different set of pieces of special equipment.

These different kinds of attribute propagation from an abstract class to its concrete class are discussed in detail in [9].

Composition. Materializations can be composed in hierarchies, where the concrete class of one materialization is also the abstract class of another materialization. For example, the materializations Play—∗Setting—∗Performance models that theater Plays materialize as Settings that embody the production decisions for a theatrical season. Settings in turn materialize as Performances, at a particular date, with each role of Play assigned to a specific actor for each Performance.

Class transitivity. Materialization is class transitive: for classes A, C, and D, materializations A—∗C—∗D imply A—∗D.

Class nonrecursivity. Materialization is class nonrecursive, in that a class cannot materialize in itself.

Materialization semantics. The semantics of materialization is defined as a combination of generalization, classification, and of a class/metaclass correspondence. This is expressed as a collection of *two-faceted* constructs, each one being a composite structure comprising an object, called the *object facet*, and an associated class, called the *class facet*. The object facet is an instance of the abstract class and the class facet is a subclass of the concrete class. Details about this semantics are given in [9].

6.2 Aggregation

Aggregation (e.g., [16,22,24,30,46]) is a binary relationship with pattern WholeClass◇—PartClass by which a set of (component) objects is considered a higher-level (aggregate) object. For example, Figure 18 shows two aggregations between composite Newspaper and components Editorial and Article.

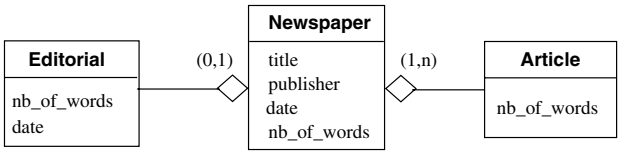


Fig. 18. Examples of aggregation

Cardinality. For the composite role, cardinality determines how many components can be grouped together to form a composite. For example, in Figure 18, a newspaper is composed of (1,n) articles. For the component role it specifies the number of composites that a component can be part of.

The lifetime of parts sometimes depends on that of their wholes and conversely. The *part-to-whole* dependency means that the existence of a part depends on the existence of the corresponding whole, i.e., that the deletion of the whole implies the deletion of the part. As an example, Journal◇—Article is part-to-whole dependent if the deletion of a journal implies the deletion of its articles. The *whole-to-part* dependency means that the existence of a whole depends on the existence of the corresponding part, i.e., the deletion of the part implies the deletion of the whole.

Attribute propagation. Some features of a whole are viewed as features of its parts and vice versa. Thus, there are two kinds of attribute propagation: *upward* propagation from the part class to the whole class and *downward* propagation from the whole class to the part class. For example, in Car◇—Body, the color of a car can be propagated upwards from its corresponding body. Similarly, in Newspaper◇—Article, the date of articles is propagated downwards from their corresponding newspaper. Furthermore, the value of a propagated attribute can be obtained as a combination of values from several source objects. For instance, the color of a car's body can be defined as some combination of the colors of its panels.

Composition. Aggregations can be composed in hierarchies, where the component class of one aggregation is also the composite class of another aggregation as in Building◇—Room◇—Wall.

Aggregation allows multiplicity for both the composite and the component roles. Figure 18 shows an example of multiplicity for the composite role. An example of multiplicity for the component role is Journal◇—Article —◇Compilation where an article can be included in a journal or in a compilation.

Class transitivity. Aggregation is not class transitive in general. For example, the aggregations Hand—◇Musician—◇Orchestra does not imply Hand—◇Orchestra. However, there are some categories of aggregations that are class transitive when taken in combination. For example, the taxonomy proposed in [46] includes seven subcategories of aggregation: (1) component—◇object (Engine—◇Car); (2) feature—◇event (Panel—◇Confe-rence); (3) member—◇collection (Advisor—◇ThesisCommittee); (4) portion —◇mass (Section—◇Chapter); (5) phase—◇activity (Analysis—◇SystemDevelopment); (6) place—◇area (City—◇Country); (7) stuff—◇object (Metal —◇Vehicle). While in [46] only aggregations belonging to the same subcategory are class transitive, class-transitive combinations of aggregations can be defined among categories (1), (4), (5), and (6) [30].

Class nonrecursivity. Aggregations can be recursive: a typical example is that of part-subpart in assemblies, where parts are composed of other parts.

Exclusiveness. Aggregations can be exclusive and shared. A shared aggregation puts no restrictions on the number of composites that a given component can be part of, allowing the component to be shared. An example is Compilation◇—Article if the same article can be included in any number of compilations.

An exclusive aggregation enforces the restriction that a given component can be part of only a single composite. Exclusiveness is natural in physical assemblies. Thus, in Car◇—Engine, two cars cannot share the same engine. This kind of exclusiveness is called *class exclusiveness* as it enforces the exclusive reference constraint within a single class. It is also the case that a car and, say, an airplane cannot share the same engine. This type of exclusiveness is called *global exclusiveness* since it bears on the entire database.

Partitioning. An aggregate class may have several partitions of component classes, each corresponding to a specific discriminator. For example School can be decomposed in two partitions: School$\overset{\text{space}}{\diamond}$—{Building,Schoolyard} and School$\overset{\text{time}}{\diamond}$—{Course, Sport}.

6.3 Role

Role [2,40,45,13,6,47,10] is a binary relationship with pattern Object-Class○←RoleClass relating an object class and a role class describing dynamic states for the object class. This definition of role should not be confused with the concept of roles in the entity-relationship model. Differences between those concepts are discussed in [6].

Figure 19 shows two role relationships relating an object class Person, and role classes Student and Employee. In a role relationship, the object class defines permanent properties of objects while each role class defines a set of properties characterizing a particular aspect in which those objects can be viewed during their lifetime. The idea is that the role relationship captures the temporal and

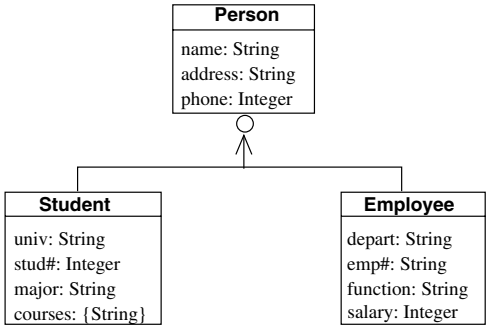


Fig. 19. Examples of roles

evolutionary aspects of real-world objects, while the usual generalization relationship deals with their static aspects. Thus, while classes Male and Female may be linked to Person via generalization links, Student and Employee would rather be linked to Person via role links. Intuitively, the role relationship offers modeling capabilities similar to generalization valid for a limited time.

Cardinality. Each instance of a role class (e.g., Student) is related to exactly one instance of its object class (e.g., Person) but, unlike generalization, each instance of the object class can be related to any number of instances of the role class, depending on the maximal cardinality at the side of the object class. For example, John can be at the same time a student in more than one university and an employee in more than one department.

Dependency. The lifetime of roles depends on that of objects playing those roles. Thus, the deletion of an object induces the deletion of its associated roles. Also, the deletion of a role may induce the deletion of its associated object if the minimal cardinality of the object class is 1. For example, if Person(1,n)○←(1,1)Employee, meaning that a person plays at least once the employee role, then the deletion of the last employee role implies the deletion of that person.

Attribute propagation. Role classes are not introduced for sharing information. This should rather be the responsibility of generalization. If Student is viewed as a subclass of Person, it inherits all properties from Person. Viewed as a role class of Person, Student does not inherit properties of Person. Instead, instances of role classes access properties of their corresponding objects by delegation.

Composition. Role relationships can be composed in hierarchies, where the role class of one role is also the object class of another role. For example, class Person may have role class Employee, and the latter may have two role classes Professor and UnitHead.

Role allows multiplicity for both the object and the role classes. Figure 19 shows an example of multiplicity for the object class. An example of multiplic-

ity for the role class is Student$\circ\!\leftarrow$Councilor$\rightarrow\!\circ$Faculty where both students and faculties can play the role of councilors in the university council.

Class transitivity. Role is class transitive: for role classes R_1, R_2, and object class O, $R_1 \rightarrow\!\circ R_2 \rightarrow\!\circ O$ implies $R_1 \rightarrow\!\circ O$.

Class nonrecursivity. Role is class nonrecursive: for each class C we cannot have $C\circ\!\leftarrow C$.

Partitioning. An object class may have several partitions of role classes. For example, in an age perspective, a person may play the role of teenager or an adult, whereas, in an employment perspective, a person may play the role of an employee or an unemployed. Therefore, there are two partitions: Person $\overset{\text{age}}{\circ\!\leftarrow}$ {Teenager, Adult} and Person $\overset{\text{employment}}{\circ\!\leftarrow}$ {Employee, Unemployed}.

6.4 Grouping

Grouping [4,33] is a binary relationship with pattern MemberClass$\rightarrow\!\!\!\!\rightarrow$ SetClass by which a collection of set members is considered as a higher-level set object. Figure 20 shows an example of grouping between the member class TennisPlayer and the set class TennisClub.

The set class in a grouping has at least one *set-determining* attribute and, optionally, a number of *set-describing* attributes and/or constraints. The set-determining attribute is the attribute whose value is the set of members. In Figure 20, the grouping class TennisClub defines three set-determining attributes grouped under the category $\langle\langle$members$\rangle\rangle$. A set-describing attribute is an attribute whose value is derived from attributes of the set of members. In Figure 20, under the category $\langle\langle$attributes$\rangle\rangle$ there are two ordinary attributes name and fee and one set-describing attribute avgAgeOfMembers holding the average age of the club members.

The most important difference between the concepts of grouping and set is that a grouping is concerned with properties and constraints of the grouping viewed as a whole in addition to set membership. Thus, whereas two sets are equal if and only if they have the same members, this is not necessarily so for groupings. Two groupings having the same members, for example, two specific

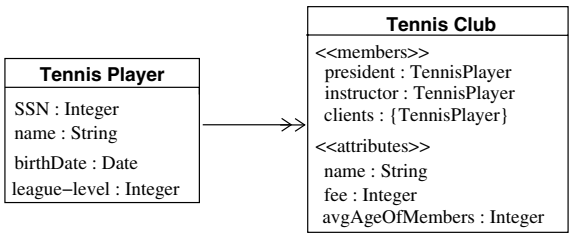


Fig. 20. An example of grouping

clubs, may differ in their internal identifiers or in the values of some property associated with the grouping, such as the minimum age required to be a member of the club.

Cardinality. In general, the grouping relationship constrains a set class to have at least one member whereas the participation of a member in the grouping may be optional or mandatory. For example, the cardinalities in Person(0,1)⇾(1,n)PoliticalParty mean that every person may be a member of at most one political party and that every political party has at least one member.

Dependency. In general, the lifetime of members does not depend on that of their groupings and conversely. However, a dependency may be implied by the cardinality constraints. For example, in Employee(1,1)⇾(1,20)De-partment, due to the cardinality (1,1), the deletion of a department implies the deletion of its employees.

Attribute propagation. Grouping can be seen as a kind of aggregation. Hence, both upwards and downwards propagation are possible, although this is not clearly stated in [4] or [33].

Composition. Grouping can be composed in hierarchies, where the member class of one grouping is also the set class of another grouping. An example is TennisPlayer⇾⇾TennisClub⇾⇾TennisFederation.

Grouping allows multiplicity for both the member and the set classes. Examples are, respectively, TennisClub⇽⇽Employee⇾⇾TradeUnion and Person⇾⇾Sponsors⇽⇽Organization.

Class transitivity. Grouping is normally not class transitive. For example, Book⇾⇾Library ⇾⇾LibraryNetwork does not imply Book⇾⇾LibraryNetwork.

Class nonrecursivity. Grouping is class nonrecursive: a class cannot be member of itself.

Exclusiveness. In the grouping relationship, members can be exclusive or shared. A shared member puts no restrictions on the number of groupings that a given element can be member of, allowing the member to be shared. For example TennisClub⇽⇽TennisPlayer is shared if the same player can be a member in any number of clubs.

Member covering. This specifies whether or not all instances of the member class are necessarily related to an instance of the grouping class.

Partial covering means that there is at least one member that does not belong to any grouping. For example, in Employee⇾⇾TennisClub, not all employees must be members of the tennis club. *Complete covering* means that the grouping provides a complete covering of the member class. For example, in TennisPlayer⇾⇾TennisClub the grouping class TennisClub covers all instances of the member class TennisPlayer.

6.5 Ownership

Ownership [48,17] is a binary relationship with pattern Property$\cdots\succ$ Owner relating an owner and a property that is possessed. For example, in Figure 21 Person is the owner class and BankAccount is the property class.

Intuitively, ownership means that the owner of a property has certain rights on the property. Various shades of ownership express the intuitive semantics of the relationship. Thus, the owner can be a person or a legal entity (e.g., a corporation or an organization). Property ownership can be temporary or permanent. A property can be *real* (e.g., a piece of land), *intellectual* (e.g., an idea, a creative work, a patent), or *personal* encompassing everything that is not a real or an intellectual property.

Cardinality. In general, ownership constrains a property to have at least one owner whereas an owner may have 0 or several properties. In the example of Figure 21, a person can have (0,n) bank accounts and a bank account can have (1,n) owners.

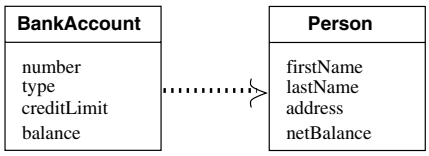


Fig. 21. An example of ownership

Dependency. The deletion of a property can cause deletion of the owner. For example, suppose that an insurance company distinguishes people who own cars from people who do not. This can be modeled by a class Person with a subclass CarOwner and an ownership CarOwner$\prec\cdots$Car. In this case, the car owner is dependent on the car, i.e., if a car owner only owns one car and this car is deleted, then the car owner should be deleted from class CarOwner (but not from class Person).

An example of dependency of the property on the owner is an ownership Employee$\prec\cdots$Car with an additional constraint stating that, when employees stop working for the company, the information about their cars is no longer needed.

Attribute propagation. Some features of a property are naturally viewed as features of its owner or vice versa. For example, the address of persons may be modeled as the address of their house rather than as an attribute of persons. Likewise, the name on a passport can be modeled as the name of the passport owner. In the former case, the value of address is propagated upwards from the property to the owner. In the latter case, the value of name is propagated downwards from the owner to the property. Furthermore, the value of the propagated

attribute can be obtained as a combination of values from several objects linked through ownership. For example, in Figure 21, attribute netBalance of a person is computed as the sum of the balance of the person's bank accounts.

Composition. Ownership can be composed where the property of an ownership is also the owner of another ownership, as in Corporation$\prec\cdots$Division $\prec\cdots$Factory.

Ownership allows multiplicity for both the property and the owner roles. Thus, owner classes can own several properties as in Vehicle$\cdots\succ$Person$\prec\cdots$House. Also, a property can be owned by several owners as in Person$\prec\cdots$Stock$\cdots\succ$Company.

Class transitivity. Ownership is generally not class transitive. A counterexample is that Person$\prec\cdots$Cat$\prec\cdots$Claw does not imply Person$\prec\cdots$Claw.

Class nonrecursivity. Ownership can be recursive: for example, a company can own other companies.

Exclusiveness. Ownership can be exclusive or *joint*, i.e., a property may be owned by one owner or shared by several owners. Person$\prec\cdots$Retirement-Portfolio is an example of exclusive ownership.

There are two types of joint ownerships. *Free joint* ownership states no explicit partition of the rights of the joint owners in the property. For example, a joint bank account is freely shared by a couple. In *percentage joint* ownership, each owner takes a percentage of the ownership, e.g., when husband and wife each owns 50% of their house. An *equal joint* is when all owners have the same percentage. As noted in [17], percentage joint is unique to ownership, while exclusiveness also concerns other generic relationships.

Partitioning. In the ownership relationship, the owner may own several categories of properties each according to a given perspective. For example, consider the owner class Person. Viewed as a biological being, a person owns a brain and a heart. Viewed as a psychological being, a person possesses a certain responsibility and a personality. We have therefore two partitions: Person $\overset{\text{biology}}{\prec\cdots}$ {Brain, Heart} and Person $\overset{\text{psychology}}{\prec\cdots\cdots}$ {Responsibility, Personality}.

7 New Generic Relationships

This section gives some guidelines to identify and define new generic relationships.

Information modeling focuses on capturing and representing certain aspects of the real world relevant to the functions of an information system. The central constructs in the building process of information models are entities (or types, classes) representing important things of the application domain, and relationships among those things.

When building an information model, it is relatively easy to identify adequate entities to capture real-world objects: they directly correspond to the important concepts naturally manipulated by stakeholders in the application domain. The research reviewed in this paper advocates the use of a rich repertoire of generic relationships for modeling relationships between entities. Thus, for the information modeler, the choice of appropriate relationships to associate objects is comparatively more difficult.

Various choices of relationships correspond to sometimes subtle differences in the shades of real-world semantics captured in an information model. For example, the relationship between books and their book copies is better modeled as a materialization than as an association.

In practice, when deciding on which relationship to use for modeling a relationship, the modeler has to choose between: (1) a usual association (like in Employee—(works)—Employer), (2) a specific relationship derived from a generic relationship within the repertoire of available generic relationships (such as the relationship Employee→○Person derived from RoleClass→○ObjectClass), and (3) a specific relationship derived from a new generic relationship identified in the application domain.

New relationships can be identified when the same pattern is repeatedly encountered and it does not fit well the available generic relationships, but the decision to define a new generic relationship should be made with care. For example, some candidate generic relationships can be best described as subcategories of already identified relationships (like the subcategories of aggregation discussed in, e.g., [24,42,46]).

When a new generic relationship has been tentatively identified, it must be well defined (i.e., it must be intuitively well understood), it should correspond to a significant number of specific instances validated in application domains, its semantics should be formalized, and it should be associated with an appropriate graphical notation.

The intuitive semantics of a generic relationship is a broad intent about the duties of the relationship in application domains, in the style of the short descriptions of Section 2. The formal semantics fits in two categories: the first one positions the relationship along the various characteristics reviewed in Section 5 while the second characterizes the inherent semantics of the relationship, in particular in terms of the semantics of creation, update, and deletion of objects involved in the relationship.

A graphical notation for a generic relationship includes a notation for participating classes and for the relationship itself. Notations of a varying degree of detail can be defined for participation constraints linked with cardinality (see, e.g., [24]).

The identification and definition of new relationships should carefully explore and characterize their similarities and interactions with existing relationships [10].

8 A Refined Metamodel for Relationships

This section presents our metamodel for generic relationships, based on their semantics as presented in Sections 4 and 5. Its general structure is shown in Figure 22. It is an elaborated version of the hierarchy in Figure 3.

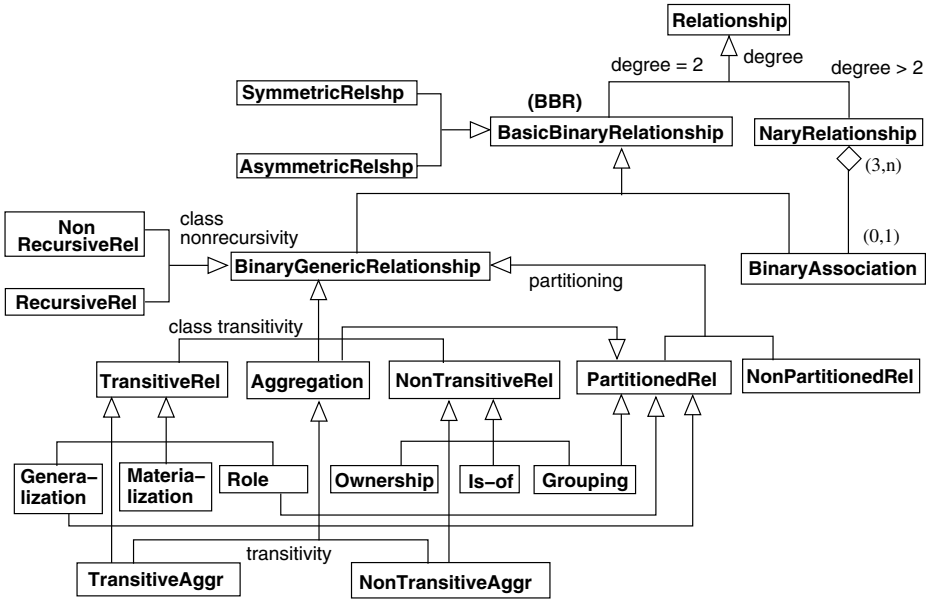


Fig. 22. Our metamodel of generic relationships

Metatype Relationship denotes all kinds of generic relationships. The first partitioning, on the top of the hierarchy, is based on the degree of relationships. We distinguish binary relationships, represented in the figure by metatype BasicBinaryRelationship, and n-ary relationships, represented by metatype NaryRelationship. BasicBinaryRelationship represents the basic binary relationship $\mathcal{BBR}$ described in Section 4. BasicBinaryRelationship is specialized in SymmetricRelshp and AsymmetricRelshp representing, respectively, symmetric and asymmetric relationships.

Metatype BasicBinaryRelationship is in turn classified in two categories: BinaryGenericRelationship representing binary generic relationships described in Section 5 and BinaryAssociation representing binary associations. This decomposition is needed for the following two reasons. First, as seen in Section 5, binary generic relationships (e.g., generalization, materialization, aggregation) can be described along dimensions such as multiplicity, composition, class transitivity, class nonrecursivity, and partitioning. These properties do not concern binary associations represented by metatype BinaryAssociation. Second, an n-ary relationship, represented by metatype NaryRelationship, is defined as an aggregate

of three or more binary associations. Note that NaryRelationship cannot be defined as an aggregate of $\mathcal{BBR}$, because the latter also accounts for binary generic relationships such as generalization, aggregations, etc., that cannot compose an n-ary relationship.

Binary generic relationships are classified along class transitivity, class nonrecursivity, and partitioning, which only apply to some generic relationships.

Class-transitive relationships are grouped under category TransitiveRel while nontransitive ones go under category NonTransitiveRel. Aggregation cannot be directly placed under TransitiveRel nor under NonTransitiveRel because some of its categories are class transitive and others are not. The former are gathered under metatype TransitiveAggr and the latter under metatype NonTransitiveAggr. TransitiveAggr and NonTransitiveAggr are defined as subclasses of TransitiveRel and NonTransitiveRel, respectively. Similarly, recursive and nonrecursive generic relationships are represented in the metamodel by metatypes RecursiveRel and NonrecursiveRel.

In parallel to the specialization of binary generic relationships along the dimensions of class transitivity and class nonrecursivity, their specialization along the partitioning dimension gives rise to two metatypes: PartitionedRel representing binary generic relationships (e.g., generalization, aggregation, role, and grouping) which may be organized in several partitions along several discriminators and NonPartitionedRel representing relationships which may not.

This metamodel can be enriched if other dimensions for classifying binary generic relationships are identified.

9 Conclusion

This paper has discussed relationships in information modeling. We first defined a basic binary relationship $\mathcal{BBR}$ that both defines binary associations and represents the common semantics of generic relationships. The semantics of $\mathcal{BBR}$ was defined along several dimensions, including cardinality, existence dependency, exclusiveness and inclusion, symmetry and asymmetry, instance transitivity, and attribute propagations.

We then defined binary generic relationships as specializations of $\mathcal{BBR}$ and characterized them along several important dimensions like class and instance semantics, multiplicity, composition, class transitivity, class nonrecursivity, and partitioning. Then, several generic relationships were reviewed in the light of those dimensions. Therefore, instead of defining the semantics of generic relationships in an ad-hoc manner as is often done, those dimensions provide a useful support for a clear and systematic definition.

We also discussed how new generic relationships can be identified and defined. A new generic relationship should correspond to a significant number of specific instances validated in application domains, its semantics should be formalized according the dimensions above, and it should be associated with an appropriate graphical notation. The identification and definition of new relationships should carefully explore and characterize their similarities with existing relationships.

Finally, we defined a new metamodel for generic relationships based on their semantics.

We are currently comparing our metamodel for relationships with that proposed in UML 2.0. The study would result in the extension of UML with new generic relationships that have no equivalent in UML such as role-of and materialization.

References

1. J. Abrial. Data semantics. In *Proc. of the IFIP Working Conf. on Data Base Management*, pages 1–59. North-Holland, 1974.
2. A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An object data model with roles. In *Proc. of the 19th Int. Conf. on Very Large Data Bases, VLDB'93*, pages 39–51. Morgan Kaufmann, 1993.
3. E. Andonoff, G. Hubert, and A. Le Parc. Modeling inheritance, composition and relationship links between objects, object versions and class versions. In *Proc. of the 7th Int. Conf. on Advanced Information Systems Engineering, CAiSE'95*, LNCS 932, pages 96–111. Springer-Verlag, 1995.
4. M. Brodie. Association: A database abstraction. In P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 583–608. North-Holland, 1981.
5. R. Cattell, D. Barry, M. Berler, and J. Eastman, editors. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
6. W. Chu and G. Zhang. Associations and roles in object-oriented modeling. In *Proc. of the 16th Int. Conf. on Conceptual Modeling, ER'97*, LNCS 1331, pages 257–270. Springer-Verlag, 1997.
7. P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors. *Proc. of the 8th Int. Conf. on Advanced Information Systems Engineering, CAiSE'96*, LNCS 1080. Springer-Verlag, 1996.
8. M. Dahchour. *Integrating Generic Relationships into Object Models Using Metaclasses*. PhD thesis, Département d'ingénierie informatique, Université catholique de Louvain, Belgium, 2001.
9. M. Dahchour, A. Pirotte, and E. Zimányi. Materialization and its metaclass implementation. *IEEE Trans. on Knowledge and Data Engineering*, 14(5):1078–1094, 2002.
10. M. Dahchour, A. Pirotte, and E. Zimányi. A role model and its metaclass implementation. *Information Systems*, 29(3):235–270, 2004.
11. K. Davis, G. Dong, and A. Heuer. Discussion report: Object migration and classification. In *Proc. of the 4th Int. Workshop on Foundations of Models and Languages for Data and Objects*, pages 223–227. Springer-Verlag, 1992.
12. D. Firesmith, B. Henderson-Sellers, and I. Graham. *OPEN Modeling Language OML Reference Manual*. SIGS Books, 1997.
13. G. Gottlob, M. Schrefl, and B. Röck. Extending object-oriented systems with roles. *ACM Trans. on Office Information Systems*, 14(3):268–296, 1996.
14. R. Gupta and G. Hall. An abstraction mechanism for modeling generation. In *Proc. of the 8th Int. Conf. on Data Engineering, ICDE'92*, pages 650–658. IEEE Computer Society, 1992.
15. G. Hall and R. Gupta. Modeling transition. In *Proc. of the 7th Int. Conf. on Data Engineering, ICDE'91*, pages 540–549. IEEE Computer Society, 1991.

16. M. Halper, J. Geller, and Y. Perl. An OODB part-whole model: Semantics, notation, and implementation. *Data & Knowledge Engineering*, 27(1):59–95, 1998.
17. M. Halper, Y. Perl, O. Yang, and J. Geller. Modeling business applications with the OODB ownership relationship. In *Proc. of the 3rd Int. Conf. on AI Applications on Wall Street*, pages 2–10, 1995.
18. T. Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann, 2001.
19. B. Henderson-Sellers. OPEN relationships: Compositions and containments. *Journal of Object-Oriented Programming*, 10(7):51–55, 1997.
20. T. Jones and I. Song. Binary equivalents of ternary relationships in entity-relationship modeling: A logical decomposition approach. *Journal of Database Management*, 11(2):12–19, 2000.
21. R. Katz. Towards a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, 1990.
22. W. Kim, E. Bertino, and J. Garza. Composite objects revisited. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, SIGMOD'89*, pages 337–347, 1989. SIGMOD Record 18(2).
23. M. Kolp. *A Metaobject Protocol for Integrating Full-Fledged Relationships into Reflective Systems*. PhD thesis, INFODOC, Université Libre de Bruxelles, Belgium, 1999.
24. M. Kolp and A. Pirotte. An aggregation model and its C++ implementation. In *Proc. of the 4th Int. Conf. on Object-Oriented Information Systems, OOIS'97*, pages 211–224, 1997.
25. Y. Lahlou and N. Mouaddib. Relaxing the instantiation link: Towards a content-based data model for information retrieval. In Constantopoulos et al. [7], pages 540–561.
26. A. Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, 24(11):908–926, 1998.
27. H. Lieberman. Using prototypical objects to implement shared behavior in object oriented systems. In *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications, OOPSLA'86*, pages 214–223, 1986. ACM SIGPLAN Notices 21(11), 1986.
28. T. Ling. A normal form for entity-relationship diagrams. In *Proc. of the 4th Int. Conf. on the Entity-Relationship Approach, ER'85*, pages 24–35, 1985.
29. N. Mattos. Abstraction concepts: The basis for data and knowledge modelling. In *Proc. of the 7th Int. Conf. on the Entity-Relationship Approach, ER'88*, pages 473–492, 1988.
30. R. Motschnig-Pitrik and J. Kaasboll. Part-whole relationship categories and their application in object-oriented analysis. *IEEE Trans. on Knowledge and Data Engineering*, 11(5):779–797, 1999.
31. R. Motschnig-Pitrik and J. Mylopoulos. Classes and instances. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):61–92, 1992.
32. R. Motschnig-Pitrik and J. Mylopoulos. Semantics, features, and applications of the viewpoint abstraction. In Constantopoulos et al. [7], pages 514–539.
33. R. Motschnig-Pitrik and V. Storey. Modelling of set membership: The notion and the issues. *Data & Knowledge Engineering*, 16(2):147–185, 1995.
34. J. Mylopoulos. Information modeling in the time of the revolution. *Information Systems*, 23(3–4):127–155, 1998.

35. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about informations systems. *ACM Trans. on Office Information Systems*, 8(4):325–362, 1990.
36. J. Odell. Six different kinds of composition. *Journal of Object-Oriented Programming*, 6(8):10–15, 1994.
37. C. Parent, S. Spaccapietra, and E. Zimányi. *Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS approach*. Springer, 2005, to appear.
38. J. Peckham, B. MacKellar, and M. Doherty. Data model for extensible support of explicit relationships in design databases. *Very Large Data Bases Journal*, 4(2):157–191, 1995.
39. A. Pirotte, E. Zimányi, D. Massart, and T. Yakusheva. Materialization: a powerful and ubiquitous abstraction pattern. In *Proc. of the 20th Int. Conf. on Very Large Data Bases, VLDB'94*, pages 630–641, 1994. Morgan Kaufmann.
40. D. Renouf and B. Henderson-Sellers. Incorporating roles into MOSES. In *Proc. of the 15th Conf. on Technology of Object-Oriented Languages and Systems, TOOLS 15*, pages 71–82, 1995.
41. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language: Reference Manual*. Addison-Wesley, second edition, 2004.
42. V. Storey. Understanding semantic relationships. *Very Large Data Bases Journal*, 2(4):455–488, 1993.
43. T. Teorey. *Database Modeling and Design*. Morgan Kaufmann, third edition, 1999.
44. J. Wäsch and K. Aberer. Flexible design and efficient implementation of a hypermedia document database system by tailoring semantic relationships. In *Proc. of the IFIP WG2.6 6th Working Conf. on Database Semantics, DS-6*, pages 367–388. Chapman & Hall, 1995.
45. R. Wieringa, W. De Jonge, and P. Spruit. Using dynamic classes and role classes to model object migration. *Theory and Practice of Object Systems*, 1(1):61–83, 1995.
46. M. Winston, R. Chaffin, and D. Herrmann. A taxonomy of part-whole relations. *Cognitive Science*, 11(4):417–444, 1987.
47. R. Wong, H. Chau, and F. Lochovsky. A data model and semantics of objects with dynamic roles. In *Proc. of the 13th Int. Conf. on Data Engineering, ICDE'97*, pages 402–411. IEEE Computer Society, 1997.
48. O. Yang, M. Halper, J. Geller, and Y. Perl. The OODB ownership relationship. In *Proc. of the Int. Conf. on Object-Oriented Information Systems, OOIS'94*, pages 278–291. Springer-Verlag, 1994.
49. E. Yu, L. Liu, and Y. Li. Modelling strategic actor relationships to support intellectual property management. In *Proc. of the 20th Int. Conf. on Conceptual Modeling, ER 2001*, pages 164–178, 2001.