A Generic Role Model for Dynamic Objects

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

[1] University of Louvain, IAG School of Management, Information Systems Unit (ISYS)
1 Place des Doyens, 1348 Louvain-la-Neuve, Belgium
dahchour@isys.ucl.ac.be
pirotte@info.ucl.ac.be

[2] University of Brussels, Informatics Department
50 Av. F. Roosevelt, C.P. 165, 1050 Brussels, Belgium
ezimanyi@ulb.ac.be

Abstract. The *role* generic relationship for conceptual modeling relates a class of objects (e.g., persons) and classes of roles (e.g., students, employees) for those objects. The relationship is meant to capture temporal aspects of real-world objects while the common generalization relationship deals with their more static aspects. This paper presents a generic role model, where the semantics of roles is defined at both the class and the instance levels. The paper also discusses the interaction between the role relationship and generalization, and it attempts to clarify some of their similarities and differences.

Keywords: Information modeling, role model, object technology

1 Introduction

Object models represent real-world applications as a collection of objects and classes. Objects model real-world entities while classes represent sets of similar objects. The *classification/instantiation* relationship relates a class to its instances. Classes are organized in *generalization/specialization* hierarchies where subclasses inherit structure and behavior from their superclasses.

Most object models assume that an object cannot be an instance of more than one class at the same time (except in the presence of generalization) and that an object cannot change its class. Those assumptions are not well suited to modeling some dynamic situations from the real world. Consider class Person specialized into classes Student and Employee. If John is created as an instance of Student, it also becomes an instance of Person. But John cannot be created as an instance of Person and later become in addition an instance of Student. Neither can student John change its class to become, say, an employee.

Some models accept *overlapping* generalizations with, e.g., subclasses Student and Employee of Person sharing instances. This leads to multiple classification, which is discussed later.

An alternative is to allow *multiple inheritance* with *intersection classes*. Thus, for example, an intersection class StudentEmployee could be created as a common subclass of Student and Employee. Multiple inheritance may lead to a combinatorial explosion in the number of subclasses. Also, an instance of StudentEmployee cannot be viewed as only a student or only an employee. Such a context-dependent access to properties is not supported by generalization [12].

Multiple classification [5] allows an object to be a direct instance of more than one class (e.g., John as an instance of both Student and Employee). Although multiple classification avoids the combinatorial explosion of multiple inheritance, it does not allow a context-dependent access to object properties: the attributes of an object comprise all the attributes of its classes. It is thus not possible, e.g., to view John either as a Student or as an Employee.

Another problem pointed out in, e.g., [12,20,24,25] is the impossibility of representing the same real-world object as more than one instance of the same class. For example, John could be a student at two universities, say ULB and UCL, with, in each of them, a student number and a registration in a number of courses. A possible modeling of such a situation requires three classes: two subclasses ULB_Students and UCL_Students of Student, and an intersection class ULB_UCL_Students as a subclass of both ULB_Students and UCL_Students. Such a solution is heavy and impractical.

The role relationship [2,7,10,12,19,24,25] was proposed as a way to overcome those limitations of classical object models. It captures evolutionary aspects of real-world objects that cannot be modeled by time-dependent attribute values and that are not well captured by the generalization relationship.

The rest of the paper is organized as follows. Section 2 presents a comprehensive semantics for our role model. Section 3 addresses some issues about object identity and role identity. Section 4 explores similarities and differences between the role and generalization relationships. Section 5 introduces two concepts for specifying role control: *meaningful combinations* of role classes, used for monitoring membership in several role classes, and *transition predicates*, used for specifying when objects may evolve by gaining and/or losing roles. Section 6 summarizes the class- and instance-level semantics of our role relationship. Section 7 discusses related work and suggests criteria to account for the variety of approaches for role modeling. Section 8 summarizes and concludes the paper.

2 Our Role Model: General Structure

This section presents the general structure of our role model. We will sometimes simply say "role-of" for the role relationship of our model.

Role-of (of pattern ObjectClass$\circ\!\!\longleftarrow$RoleClass) relates a class, called *object class*[1], and another class, called *role class*, that describes dynamic roles for the object class. Figure 1 shows two role relationships relating the Person object class to role classes Student and Employee. We say that instances of the object class gain roles (also that they *play* roles), which are instances of the role class. The

[1] The *object class* of a role relationship has sometimes been called *root type*, *base class*, *player class*, *base role*, *natural type*, or *role model*.

object class defines permanent properties of objects over their lifetime, while each role class defines some of their transient properties. When the context is clear, instances of object classes will be called *objects* while instances of role classes will be called *roles*.

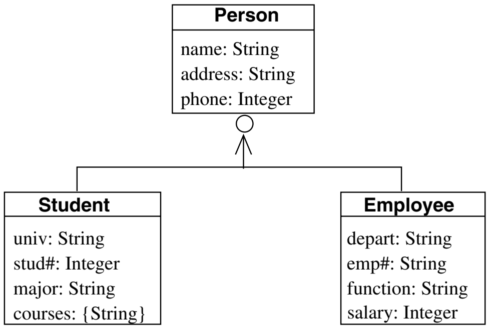

Fig. 1. An example of role relationship

The role concept addresses three main issues arising when modeling evolving entities with traditional object models:

(1) **Dynamic change of class**, i.e., objects changing their classification. If the object undergoing the transition remains an instance of the source class, the transition is called an **extension**. Otherwise, it is called an **evolution**.
(2) **Multiple instantiation of the same class**, i.e., an object becoming an instance more than once of the same class, while losing or retaining its previous membership. For example, a student may be registered in two different universities.
(3) **Context-dependent access**, i.e., the ability to view a multi-faceted object in a particular perspective. For example, person John can be viewed separately either as an employee or as a member of the golf club.

A role can be viewed as a possible state for an object. An evolving object is represented by an instance of an object class and by a set of instances of role classes. The basic idea is that the set of instances of role classes for the object can evolve easily over time. Still, some state changes are naturally modeled as changes of attribute values rather than as roles. The idea is that roles concern new responsibilities, facets, or aspects. For example, a salary raise could be naturally modeled as the change of an attribute value, while an employee becoming a member of a golf club could be naturally modeled as a new role for the employee.

Some versions of the entity-relationship (ER) model associate a notion of role with relationships. An entity involved in a relationship is said to play a role. Most often those roles are little more than an alternative name for the relationship, emphasizing the viewpoint of one participating entity. According to [7], a role

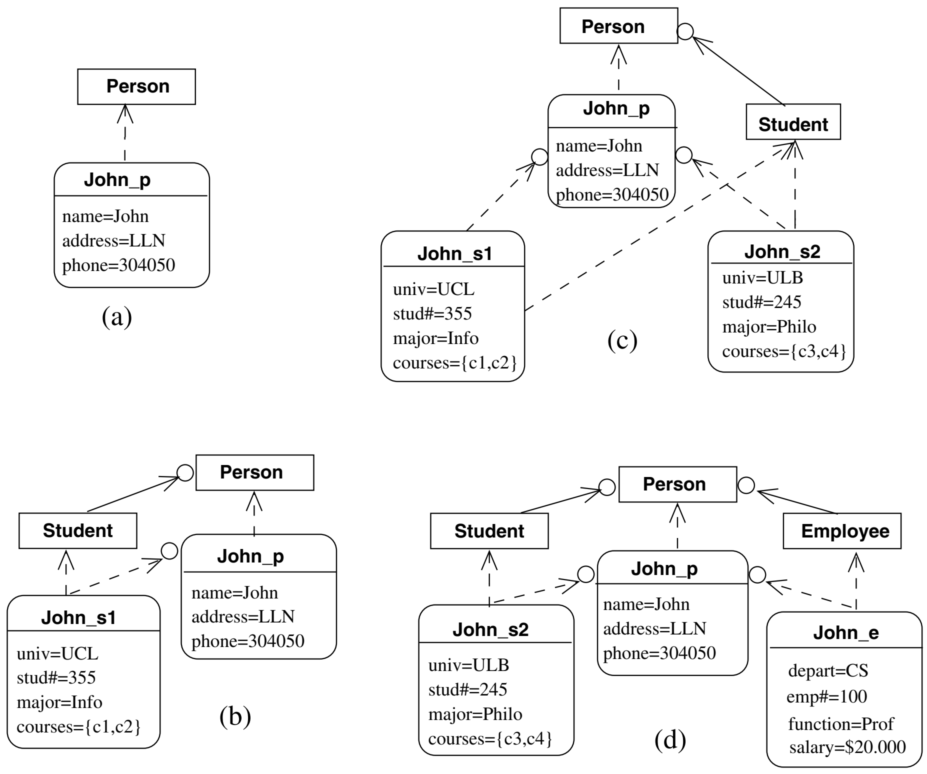

Fig. 2. Various roles for an object

class $\mathcal{R}_1$ related to an object class $\mathcal{O}$ (i.e., $\mathcal{R}_1 \rightarrow\circ \mathcal{O}$) necessarily participates as a role (in the sense of ER model) in a specific relationship. For example, the intuition behind the roles of Figure 1 is that persons may play the role of students, registered in some universities, and of employees, working in some departments. Thus, classes Student and Employee can be viewed as role classes in relationships Student$\overset{registers}{\text{———}}$University and Employee$\overset{works}{\text{———}}$Department, respectively. Of course, the semantics underlying the role concept of relationship role-of cannot be wholly captured by the role concept of the ER model.

Figure 2[2] shows on some instances of the schema of Figure 1 how persons may evolve by gaining and/or losing roles. In Figure 2(a), John_p is created as instance of Person. Figure 2(b) shows an instance John_s1 of Student, related to John_p by the role relationship, expressing that John_p has become a stu-

[2] We draw classes as rectangular boxes and instances as boxes with rounded corners. Classification links appear as dashed arrows and generalization links as solid arrows. Role-of relationships at the class level are drawn as solid oriented links with a circle on the side of the object class. Role-of links at the instance level are drawn as dashed oriented links with a circle on the side of the object instance.

dent. Both instances John_p and John_s1 coexist with different identifiers (see Section 3). If John ceases to be a student, then instance John_s1 will just be removed. Figure 2(c) shows another instance of Student, John_s2, modeling that John has registered at another university (ULB). Figure 2(d)shows John having left the UCL, become an employee (John_e), while still being a student at the ULB university.

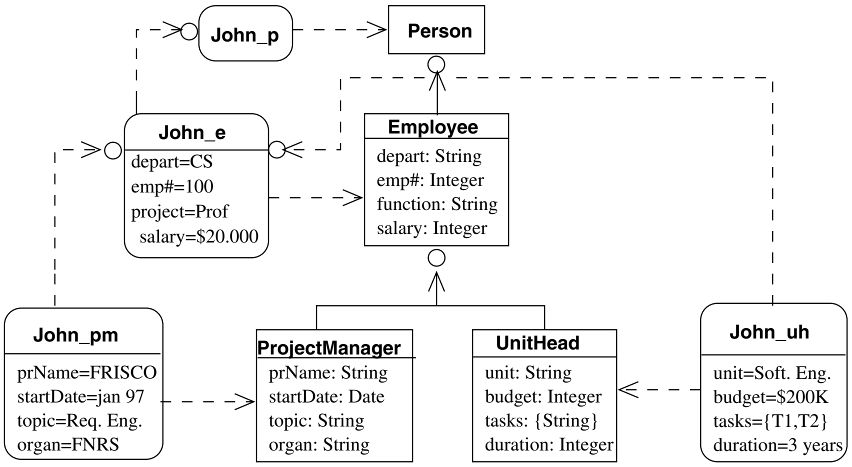

Fig. 3. Composition of roles

Role relationships can be composed in hierarchies, where the role class of one role relationship is also the object class of another role relationship, like Employee in Figure 3, which is a role class for class Person and an object class for role classes ProjectManager and UnitHead. Persons can be ProjectManager only if they play the role of Employee. Figure 3 also shows John_e of Figure 2(d) as an object with a new role John_pm, instance of ProjectManager. If John is promoted as head of the software-engineering unit at the CS department, this role is represented as an instance (John_uh) of UnitHead. John_e is thus a role for John_p, and an object for John_pm and John_uh.

3 Object Identity versus Role Identity

The concept of object identity has a long history in programming languages [13]. It is more recent in database management [14]. In a model with object identity, an object has an existence which is independent of its attribute values. Hence identity (twice the same object) is not the same thing as equality (objects with the same attribute values)[3].

A related concept called *role identity* (rid) is introduced and compared to object identity in [23]. The need for both oid and rid is motivated in [24] as follows:

assume that Passenger is a subclass of Person (in the sense of generalization) and consider persons who migrate to Passenger, say, by boarding a bus. A bus may carry 500 passengers during one day, but these passengers may actually be only 100 different persons, if a person may board the bus several times. Hence the necessity of two ways of counting passengers.

The migration of an object from a source class to target class(es), with the object losing or retaining its previous membership, introduces the issue of object-identity conservation, i.e., whether the object changes its identifier in the case that it loses its previous membership and whether it retains its identifier in the case that it remains an object of the source class.

In the literature, one approach opts for oid conservation (e.g, [15,16,22]), while another identifies roles independently of their associated objects, using the notion of role identifiers (e.g., [10,12,23]). For example, consider person John who becomes a student and then an employee, while remaining a student. In the first approach, John is represented by a single oid, while in the second, John is represented by one oid, that of the first state as a Person, and two independent rids corresponding to John as a student and as an employee, respectively.

The first approach presents several disadvantages. First, it fails to capture objects that can have several role objects of the same class, like John as a student at two different universities. The second approach easily handles that case by creating several instances of the same role class, each with a different rid. Second, changing the oid of John into that of a student object and in turn into that of an employee object raises the issue of dangling references, unlike the second approach, which represents each new role instance with its own rid. Thirdly, the first approach does not permit to represent historical information as the history of oids is lost. For those reasons, our model follows the second approach.

4 Roles versus Generalization

This section compares the role and generalization relationships. Section 4.1 examines how generalization alone approaches object dynamics. Section 4.2 discusses some differences between the relationships and Section 4.3 presents various cases where a class may be involved in both generalization and role relationships.

4.1 Dealing with Object Dynamics with Generalization Alone

Dealing with object dynamics with generalization is heavy and impractical [1], as illustrated by the following examples, where the role relationships Student→∘Person∘←Employee of Figure 4 (a) are modeled as generalization Student—▷Person◁—Employee. Then:

- *how do persons become students?* If John is an instance of Person, establishing it as an instance of Student requires replacing instance John by a new instance John_s of Student and changing the references to John into references to John_s.

- *how can persons be both students and employees at the same time?* A common subclass of Student and Employee is needed (with multiple inheritance). Anticipating each meaningful combination of classes is impractical.
- *how can persons be employees at more than one department?* To represent John as an employee at two departments, three subclasses of Employee are needed, one for each of the departments and a common intersection class.

4.2 Differences between Generalization and the Role Relationship

This section points out some differences between generalization and the role relationship, illustrated in Figure 4. Some of these differences are discussed in [24].

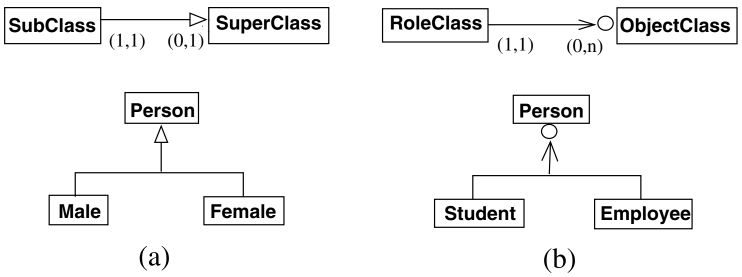

Fig. 4. Generalization (a) versus role relationship (b)

- **Cardinalities**. Each instance of a subclass (e.g., Male) is related to exactly one instance of its superclass (e.g., Person) and each instance of the superclass is related to at most one instance of its subclasses. 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 (given by the maximal cardinality n at the side of the object class).
- **Object identity.** An instance of a subclass has the same object identifier oid as the associated instance of the superclass (it is the same object viewed differently). Each instance of a role class has its own role identifier rid, different from that of all other instances of the role class. For example, the identity of student John is different from that of person John. If John is registered at two universities, there is one person instance with its oid and two student instances each with their rid.
- **Change of classes.** In most object models, an instance of superclass A that is not an instance of subclass B cannot later become an instance of B. Instead, an instance of object class A can later become or cease to be an instance of role class B. For example, an instance of Person that is not a Student can become a Student.

- **Change of subclasses.** An instance of a subclass in a partition of the superclass cannot later become an instance of another subclass of the same partition. For example, in Figure 4(a), an instance of Male cannot change to an instance of Female. Instead, an instance of a given role class in the partition of the object class can become an instance of another role class of the partition. For example, in Figure 4(b), an instance of Student can become an instance of Employee.
- **Set of instances.** When a subclass changes the set of its instances by creating new objects, then its superclass also changes its instances. For example, the creation of a Male also creates a Person. Instead, when a role class creates a new role, then the related object class does not change its instances. For example, the creation of a new role for person John (e.g., John becomes an employee or registers at a university) does not affect the instances of Person.
- **Direct versus indirect instances.** Superclasses in generalizations need not have direct instances, only their superclasses do. There is no analog to those "abstract classes" with the role relationship.
- **Inheritance.** While subclasses inherit all properties and methods of their superclass, role classes do not inherit from their object class. Instead, instances of role classes access properties and methods of their corresponding objects with a delegation mechanism. For example, consider the role relationship Student$\circ\!\!\leftarrow$Person and the two instances John_p and John_s1 of Figure 2(b). If method getPhone() is defined in class Person (and not in Student) to access the value of the phone attribute, then the message John_s1$\rightarrow$getPhone() sent to student John_s1 will be delegated to object John_p.

4.3 Combination of Generalization and Role-of

A role class $\mathcal{R}_1$ can be subclassed by other role classes $\mathcal{R}_2$ and $\mathcal{R}_3$ (i.e., $\mathcal{R}_2 \!\longrightarrow\!\!\triangleright \mathcal{R}_1 \triangleleft\!\!\longleftarrow \mathcal{R}_3$), with the usual semantics of generalization (e.g., an instance of $\mathcal{R}_2/\mathcal{R}_3$ is also an instance of $\mathcal{R}_1$, an instance of $\mathcal{R}_2$ cannot become an instance of $\mathcal{R}_3$). Instead, for role relationships $\mathcal{R}_2 \rightarrow\!\circ \mathcal{R}_1 \circ\!\leftarrow \mathcal{R}_3$, an instance of $\mathcal{R}_2$ is not an instance of $\mathcal{R}_1$ and an instance of $\mathcal{R}_2$ can become an instance of $\mathcal{R}_3$.

Two derivation rules are associated with the combination of generalization and role-of:

(1) if $\mathcal{R}_1$ is a role class of $\mathcal{O}$ and $\mathcal{R}_2$ is a role subclass of $\mathcal{R}_1$, then $\mathcal{R}_2$ is also a role class of $\mathcal{O}$ (i.e., $\mathcal{R}_2 \!\longrightarrow\!\!\triangleright \mathcal{R}_1 \rightarrow\!\circ \mathcal{O} \Rightarrow \mathcal{R}_2 \rightarrow\!\circ \mathcal{O}$);
(2) if $\mathcal{R}_1$ is a role class of $\mathcal{O}_1$ and $\mathcal{O}_1$ is a subclass of $\mathcal{O}_2$, then $\mathcal{R}_1$ is a role class of $\mathcal{O}_2$ (i.e., $\mathcal{R}_1 \rightarrow\!\circ \mathcal{O}_1 \!\longrightarrow\!\!\triangleright \mathcal{O}_2 \Rightarrow \mathcal{R}_1 \rightarrow\!\circ \mathcal{O}_2$).

Figure 5 illustrates interactions between generalization and role relationships. Class Person is subclassed by classes Male and Female. Class Male has a role class Draftee accounting for military duties. According to rule (2) above, Draftee is also a role class of Person. Class Person is refined by two role classes Student and Employee. Student is subclassed by role classes ForeignStudent and CountryStudent (an instance of ForeignStudent cannot become an instance of CountryStudent nor conversely). According to rule (1), ForeignStudent and CountryStudent in turn become role classes of class Person. Class Employee has two role classes, Professor and UnitHead. This is an example of compositions of roles.

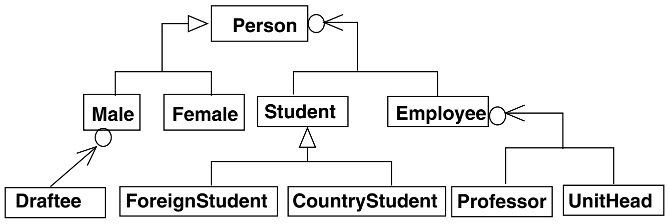

Fig. 5. Hierarchies of is-a and role-of

5 Role-Control Specification

An object can be related to several roles. This section introduces *meaningful combinations* of roles, for monitoring membership in various role classes, and *transition predicates*, for controlling how objects may gain and/or lose roles.

5.1 Meaningful Combinations of Roles Classes

When an object class $\mathcal{O}$ is related to role classes $\mathcal{R}_1, \ldots, \mathcal{R}_n$, not all combinations of roles are permitted for $\mathcal{O}$ objects. For example, if Person has roles Employee and Retired, only employees can acquire the role of retirees, but not the converse.

The notion of *meaningful combination* of role classes helps characterize the legal combinations of roles for an object. We define four types of *role combinations*:

- **Evolution**, noted $\mathcal{R}_1 \xrightarrow{ev} \mathcal{R}_2$ for role classes $\mathcal{R}_1$ and $\mathcal{R}_2$, states that an object with role r_1 of $\mathcal{R}_1$ may lose r_1 and gain a new role r_2 of $\mathcal{R}_2$, with $\mathcal{R}_1 \neq \mathcal{R}_2$. Evolutions may be bidirectional (e.g., Employee$\xleftrightarrow{ev}$Unemployed) or unidirectional (e.g., Employee$\xrightarrow{ev}$Retired), depending on whether the lost role of $\mathcal{R}_1$ may be recovered later.
- **Extension**, noted $\mathcal{R}_1 \xrightarrow{ext} \mathcal{R}_2$, with $\mathcal{R}_1$ not necessarily different from $\mathcal{R}_2$, states that an object with role r_1 of $\mathcal{R}_1$ may gain a new role r_2 of $\mathcal{R}_2$, while retaining r_1. Examples include Professor$\xrightarrow{ext}$DepartHead and Student$\xrightarrow{ext}$Student, the latter for a person allowed to register more than once as a student, for example in two different universities.
- **Acquisition**, noted $\rightarrow \mathcal{R}$, states that an object may freely acquire a role of $\mathcal{R}$, if it does not have one. For example, $\rightarrow$Student means that persons can become students. Notice that $\rightarrow$Student could be noted as Person$\xrightarrow{ext}$Student, where Person is the object class of role class Student.

- **Loss**, noted $\mathcal{R}\rightarrow$, specifies that an object may freely lose roles of $\mathcal{R}$. For example, $\mathsf{Student}\rightarrow$ means that students may cease to be students. $\mathsf{Student}\rightarrow$ could be noted as $\mathsf{Student}\xrightarrow{ev}\mathsf{Person}$.

5.2 Transition Predicates

Transition predicates can restrict or force a transition $\mathcal{R}_1\xrightarrow{mode}\mathcal{R}_2$, as illustrated in the following examples:

- The predicate $\mathsf{age} \geq 55 \wedge \mathsf{workDuration} \geq 20$ on $\mathsf{Employee}\xrightarrow{ev}\mathsf{Retired}$ states a (necessary and/or sufficient) condition for employees to retire.
- The predicate $\mathsf{contractExpirDate} \leq \mathsf{now}$ on $\mathsf{Employee}\xleftrightarrow{ev}\mathsf{Unemployed}$ specifies that employees whose contract has expired automatically become unemployed. Another predicate could state that unemployed persons with a new contract automatically become employees.
- The predicate $\mathsf{grade} \geq \mathsf{min\ and\ funding} = \mathsf{OK}$ on $\mathsf{Master}\xrightarrow{ext}\mathsf{PhD}$ specifies a condition for master students to be allowed to register as PhD students.
- The predicate $\mathsf{nbProgs} \leq \mathsf{max}$ on $\mathsf{Student}\xrightarrow{ext}\mathsf{Student}$ states that the number of registrations for a student is limited by some maximum.
- The predicate $\mathsf{age} \geq 18$ on $\rightarrow\mathsf{Employee}$ states that only adults can become employees.

6 Summary of Role Semantics

Like most generic relationships [8,9], the semantics of role-of concerns both the class and the instance levels. In the literature, these levels have mostly been considered as a whole, which has resulted in unwarranted complexity in the semantic definition and implementation of roles.

6.1 Class-Level Semantics

- An object class can have several role classes.
- A role class has exactly one object class.
- An object class has an arbitrary cardinality (c_{min}, c_{max}) regarding each of its role classes.
- Each role class has cardinality (1,1) regarding its object class.
- A role class can have other role classes.
- Meaningful combinations of role classes can be associated with object classes to provide what we have called *role control*.
- A predicate can be associated with object/role classes to describe constraints on how objects may evolve, as well as on how objects may or must automatically gain and lose roles.

6.2 Instance-Level Semantics

- A role r is not identical to an object o with that role, but it is a state in which object o can be.
- A role inherits properties and methods from its associated object by delegation.
- An object can gain several roles of the same role class or of different role classes.
- For each role, there is only one object with that role.
- Roles can be acquired and lost independently of each other.
- A role has its own identifier that is different from that of the object with the role.
- Roles evolve independently from each other or under the control of specified transition predicates.
- If an object class $\mathcal{O}$ has two role classes $\mathcal{R}_1$ and $\mathcal{R}_2$ and o is an instance of $\mathcal{O}$ with role r1 of $\mathcal{R}_1$, then r1 can only evolve as an instance of $\mathcal{R}_1$ or $\mathcal{R}_2$.
- In a composition of role-of, say $\mathcal{R}_1 {\rightarrow}\!\circ \mathcal{R}_2 {\rightarrow}\!\circ \mathcal{O}$, an object of $\mathcal{O}$ can acquire a role of $\mathcal{R}_1$ unless it already holds a role of $\mathcal{R}_2$.

7 Related Work

The interest for capturing the roles of an object goes back to the late 70's [4]. Since then, various approaches to role modeling have been proposed. We suggest the following criteria to account for the variety of those approaches.

Generalization and role-of. Various extensions to generalization were defined to deal with evolving objects (e.g., [15,16,17]). In [16], each class is considered as a particular *perspective* for its instances and the class itself denotes a role for its instances. Classes are endowed with special methods to manage roles: add a new role for an existing object, remove an existing role, etc. In [15], the concepts of *base class* and *residential class* denote, respectively, the class where an object is initially created and the class where the object currently resides; they correspond, respectively, to the *object class* and the *role class* of our model. The role model of [17] deals with evolving objects as follows. Classes are organized along the generalization hierarchy assumed to be static. If an instance of class C evolves to another class D, then D must necessarily be created dynamically as a subclass or superclass of C. D is referred to as a *role-defining* class. As a result, the application schema can be seen as composed of a static generalization hierarchy and a collections of dynamic ones.

The role model of [2] defines a single hierarchy of *role types* along the subtyping relationship, where a role subtype can have several supertypes. Thus, all classes are viewed as potentially dynamic, in that their instances may evolve over time. In that model an object is not manipulated directly but only through its roles. The answer of an object to a message depends on the role that receives it: an object does not exist on its own but is incarnated in a particular role.

In other approaches, both generalization and a specific role relationship coexist with orthogonal semantics [12,22,24,25]. In [24], the relationship between a role class and the root class (called object class) is called *played-by* (i.e., our role relationship). Role classes in turn can be refined as other role classes, and so on. The model of [12] supports both hierarchies under the names of *class hierarchy* and *role hierarchy*. A role type and its direct ancestor in the role hierarchy are linked by a relationship called *role-of*. In the DOOR model of [25], the role hierarchy is organized along the *player* relationship. The model of [22] also supports both hierarchies called *type hierarchy* and *role hierarchy*. The type hierarchy is composed of the so-called *natural* types (i.e., our object types) organized along the *subtype/supertype* relationship. The role hierarchy is composed of the so-called *roles* (i.e., our role classes) organized along the *subrole/superrole* relationship. The *role-filler* relationship (i.e., our role relationship) is defined between natural types and roles.

Preserving the oid. The issue here is whether an object that evolves over time by gaining new roles maintains its identity (i.e., oid) or acquires other identities. As mentioned in Section 3, one approach consists in preserving the single object identity regardless of how objects evolve. It is adopted by models such as [2,15,16,17] that clearly distinguish generalization and roles. Another approach argues that identifying "ordinary" objects and identifying roles raise distinct issues. As seen in Section 3, we may count one person where we count five passengers at different times of the day, which corresponds to various states of the same person. This approach has been adopted by most models, such as [10,12,24], that clearly distinguish generalization and role-of hierarchies. The approach of [25] differs from [12,23] in that roles are identified by the names of their role classes as well as their values, instead of using globally unique object/role identifiers.

Inheritance versus delegation. In addition to structure and behavior supplied by role classes, roles have to access the properties of their objects. Here again, two approaches coexist: attribute inheritance via generalization (class-based inheritance) and value propagation by delegation (instance-based inheritance). Approaches that do not clearly separate generalization and the role-of hierarchies retain attribute inheritance (e.g., [2,15,16,17]), while those that clearly distinguish them (e.g., [10,12,22,24]) mainly use delegation. Systems that follow a prototype-based paradigm (e.g., [21]) systematically use delegation.

Several roles of the same class per object. This possibility (e.g., of the same person being a student at two different universities at the same time), is easily handled in [12,24] thanks to the coexistence of both object identity (oid) and role identity (rid). Models that use the unique-oid mechanism to identify both objects and their roles (e.g., [2,15,16,17]) do not support that possibility. The model of [25] that represents roles differently than [12,24] also fails here, whereas models such as [18,20] provide the facility, although they do not introduce an explicit rid.

Context-dependent access. This means the ability to view a multi-faceted object in each role separately. For example, person John can be viewed as an employee or as a student. Most role models provide this ability. A remarkable exception is Iris [11], which allows an object to belong to several types, and to gain or lose type memberships over time, but requires the structure and methods of all the types of an object to be visible in every context. As a side effect, the name of attributes and methods must be unique across all roles.

Vertical versus horizontal role possession. This issue concerns models where role support is integrated with generalization. With vertical role possession (e.g., [17]), an object can possess roles only within the direct or indirect subclasses or superclasses of its defining class. With horizontal role possession, objects are also allowed to gain roles in so-called *sibling* or *cousin* classes (e.g., [16]). Two sibling classes are not related by a subclassing relationship nor do they have a common subclass.

Composition of roles. This refers to the possibility for a role class to serve as object class for other role classes. It holds for all models (e.g., [12,22,24,25]) that clearly separate generalization and role-of hierarchies, but not for models where the distinction is less sharp (e.g., [2,15,16,17,18]).

Role compatibility. This is the ability to explicitly control the compatibility of role possessions. For example, persons may evolve from being employees to being retirees but cannot be employees and retirees at the same time. Few models provide such a facility. The concept of *player qualification* of a role class is introduced in [25] to specify a set of classes whose instances are qualified to possess roles of the class. The concepts of *independent roles* and *coordinated roles* are introduced in [18]. Coordinating between various roles of an object is achieved through rules. The concept of *disjoint predicate* is introduced in [16] to specify that no object may be member of more than one class from a specified collection of role classes at the same time. For example, the predicate disjoint(Child, Employee, Retired) can be attached to class Person. Migration control is proposed in [15] through so-called *migration permissions*, like *relocation* (e.g., <Single, Married, relocation>) and *propagation* (e.g., <Professor, Consultant, propagation>). Relocation and propagation permissions correspond to our modes of role possession, *evolution* and *extension*, respectively. Our role-control mechanism was inspired from the migration control of [15] and enriched by the transition predicates, not supported in [15].

Transition predicates. They specify when objects may explicitly or automatically gain or lose roles from given classes. This is achieved through rules in [18]. The population of the predicate classes of [6] is governed by membership predicates for the instances of the classes. Role classes can be created in [17] by means of predicates called *role-generating* conditions, but there are no transition predicates. For example, two predicate-based classes HighlyPaidAcademic and ModeratelyPaidAcademic can be defined as subclasses of Academic by associating the predicate "Academic.salary $\geq$ 100K" to the first class and "100K $>$Academic.salary $>$ 50K" to the second class. However, there is no way to define a transition predicate $\mathcal{P}$ that would control the migration from ModeratelyPaidAcademic to HighlyPaidAcademic.

The concept of *category class* was introduced in [16] to control the *implicit* or *explicit* membership in a class with a predicate. The concept of transition predicates was inspired from [16]. However, our transition predicates have different semantics concerning the predicates associated with category classes. Given a meaningful combination of roles $\mathcal{R}_1 \xrightarrow{mode} \mathcal{R}_2$, a predicate specifies when an object with a role of $\mathcal{R}_1$ may or must automatically acquire a role of $\mathcal{R}_2$, depending on *mode*. For role relationships Teenager$\rightarrow\circ$Person$\circ\leftarrow$Adult, if predicate "age$\geq$18" is associated with the role class Adult, then Adult becomes a category class in the sense of [16], meaning that the membership in Adult depends on satisfying "age$\geq$18". We would rather associate the transition predicate "age$\geq$18" with the evolution (i.e., role combination) Teenager$\xrightarrow{ev}$Adult.

8 Conclusion

This paper has presented a generic model for roles that builds upon existing models (mostly [12,24]) and extends them in several respects (like role-control support). It clearly distinguishes between generalization and role hierarchies and supports their interaction. It also features the following characteristics:

- the possibility for roles themselves to gain roles
- the coexistence of object identifiers and roles identifiers
- a context-dependent access to objects with several roles
- the ability for an object to possess several roles of the same class
- the ability for an object to gain a role in any role class, provided the role is declared as a valid destination for the evolving object
- the coexistence of class-based inheritance (via generalization) and instance-based inheritance (delegation)
- vertical and horizontal role possession
- the control of role compatibility, through the concepts of meaningful combinations of roles and transition predicates.

References

1. L. Al-Jadir and M. Léonard. If we refuse the inheritance... In T.J.M. Bench-Capon, G. Soda, and A.M. Tjoa, editors, *Proc. of the 10th Int. Conf. on Database and Expert Systems Applications, DEXA'99*, LNCS 1677, Florence, Italy, 1999. Springer-Verlag. 648
2. A. Albano, R. Bergamini, G. Ghelli, and R. Orsini. An object data model with roles. In R. Agrawal, S. Baker, and D. Bel, editors, *Proc. of the 19th Int. Conf. on Very Large Data Bases, VLDB'93*, pages 39–51, Dublin, Ireland, 1993. Morgan Kaufmann. 644, 653, 654, 655

3. M. Atkinson, F. Bancilhon, D. Dewitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas, and S. Nishi, editors, *Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases, DOOD'89*, pages 223–240, Kyoto, Japan, 1991. North-Holland. Reprinted in the O2 Book, pp. 3–20. 647
4. C. W. Bachman and M. Daya. The role concept in data models. In *Proc. of the 3rd Int. Conf. on Very Large Data Bases, VLDB'77*, pages 464–476, Tokyo, Japan, 1977. IEEE Computer Society and ACM SIGMOD Record 9(4). 653
5. E. Bertino and G. Guerrini. Objects with multiple most specific classes. In W.G. Olthoff, editor, *Proc. of the 9th European Conf. on Object-Oriented Programming, ECOOP'95*, LNCS 952, pages 102–126, Aarhus, Denmark, 1995. Springer-Verlag. 644
6. C. Chambers. Predicate classes. In O. Nierstrasz, editor, *Proc. of the 7th European Conf. on Object-Oriented Programming, ECOOP'93*, LNCS 707, pages 268–296, Kaiserslautern, Germany, 1993. Springer-Verlag. 655
7. W. W. Chu and G. Zhang. Associations and roles in object-oriented modeling. In D.W. Embley and R.C. Goldstein, editors, *Proc. of the 16th Int. Conf. on Conceptual Modeling, ER'97*, LNCS 1331, pages 257–270, Los Angeles, California, 1997. Springer-Verlag. 644, 645
8. M. Dahchour. *Integrating Generic Relationships into Object Models Using Metaclasses*. PhD thesis, Department of Computing Science and Engineering , University of Louvain, Belgium, March 2001. 652
9. M. Dahchour, A. Pirotte, and E. Zimányi. Materialization and its metaclass implementation. Technical Report YEROOS TR-9901, IAG-QANT, Université catholique de Louvain, Belgium, February 1999. To be published in IEEE Transactions on Knowledge and Data Engineering. 652
10. N. Edelweiss, J. Palazzo de Oliveira, J. Volkmer de Castilho, E. Peressi, A. Montanari, and B. Pernici. T-ORM: Temporal aspects in objects and roles. In *Proc. of the 1st Int. Conf. on Object Role Modeling, ORM-1*, 1994. 644, 648, 654
11. D. H. Fishman, D. Beech, H. P. Cate, E. C. Chow, T. Connors, J. W. Davis, N. Derrett, C.G. Hoch, W. Kent, P. Lyngbæk, B. Mahbod, M-A. Neimat, T. A. Ryan, and M-C. Shan. IRIS: An object-oriented database management system. *ACM Trans. on Office Information Systems*, 5(1):48–69, 1987. Also in *Readings in Object-Oriented Database Systems*, Morgan-Kaufmann, 1990. 655
12. G. Gottlob, M. Schrefl, and B. Röck. Extending object-oriented systems with roles. *ACM Trans. on Office Information Systems*, 14(3):268–296, 1996. 644, 648, 654, 655, 656
13. W. Kent. A rigorous model of object reference, identity, and existence. *Journal of Object-Oriented Programming*, 4(3):28–36, June 1991. 647
14. S. N. Khoshafian and G. P. Copeland. Object identity. In N.K. Meyrowitz, editor, *Proc. of the Conf. on Object-Oriented Programming Systems, Languages and Applications, OOPSLA'86*, pages 406–416, Portland, Oregon, 1986. ACM SIGPLAN Notices 21(11), 1986. 647
15. Q. Li and G. Dong. A framework for object migration in object-oriented databases. *Data & Knowledge Engineering*, 13(3):221–242, 1994. 648, 653, 654, 655
16. E. Odberg. Category classes: Flexible classification and evolution in object-oriented databases. In G. Wijers, S. Brinkkemper, and T. Wasserman, editors, *Proc. of the 6th Int. Conf. on Advanced Information Systems Engineering, CAiSE'94*, LNCS 811, pages 406–420, Utrecht, The Netherlands, 1994. Springer-Verlag. 648, 653, 654, 655, 656

17. M. P. Papazoglou and B. J. Krämer. A database model for object dynamics. *Very Large Data Bases Journal*, 6:73–96, 1997. 653, 654, 655
18. B. Pernici. Objects with roles. In *Proc. of the Conf. on Office Information Systems*, pages 205–215, Cambridge, MA, 1990. 654, 655
19. D. W. Renouf and B. Henderson-Sellers. Incorporating roles into MOSES. In C. Mingins and B. Meyer, editors, *Proc. of the 15th Conf. on Technology of Object-Oriented Languages and Systems, TOOLS 15*, pages 71–82, 1995. 644
20. J. Richardson and P. Schwarz. Aspects: Extending objects to support multiple, independent roles. In J. Clifford and R. King, editors, *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, SIGMOD'91*, pages 298–307, Denver, Colorado, 1991. SIGMOD Record 20(2). 644, 654
21. E. Sciore. Object specialization. *ACM Trans. on Office Information Systems*, 7(2):103–122, 1989. 654
22. F. Steimann. On the representation of roles in object-oriented and conceptual modeling. *Data & Knowledge Engineering*, 35(1):83–106, October 2000. 648, 654, 655
23. R. J. Wieringa and W. de Jonge. The identification of objects and roles: Object identifiers revisited. Technical Report IR-267, Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam, December 1991. 647, 648, 654
24. R. J. 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. 644, 647, 649, 654, 655, 656
25. R. K. Wong, H. L. Chau, and F.H. Lochovsky. A data model and semantics of objects with dynamic roles. In A. Gray and P.-A. Larson, editors, *Proc. of the 13th Int. Conf. on Data Engineering, ICDE'97*, pages 402–411, Birmingham, UK, 1997. IEEE Computer Society. 644, 654, 655