ELSEVIER

Data & Knowledge Engineering 35 (2000) 83–106

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

On the representation of roles in object-oriented and conceptual modelling

Friedrich Steimann

Institut für Technische Informatik, Rechnergestützte Wissensverarbeitung, Universität Hannover, Appelstraße 4, D-30167 Hannover, Germany

Received 30 September 1999; received in revised form 10 April 2000; accepted 11 May 2000

Abstract

The duality of objects and relationships is so deeply embedded in our thinking that almost all modelling languages include it as a fundamental distinction. Yet there is evidence that the two are naturally complemented by a third, equally fundamental notion: that of roles. Although definitions of the role concept abound in the literature, we maintain that only few are truly original, and that even fewer acknowledge the intrinsic role of roles as intermediaries between relationships and the objects that engage in them. After discussing the major families of role conceptualizations, we present our own basic definition and demonstrate how it naturally accounts for many modelling issues, including multiple and dynamic classification, object collaboration, polymorphism, and substitutability. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Roles; Role-oriented modelling; Multiple and dynamic classification; Polymorphism

1. Introduction

The English merchant Lodwick (1619–1694) [56] was one of the first to break with the Aristotelian tradition according to which the nouns of a language govern its structure and meaning [19]. In his quest for a universal language he devised a system of action patterns populated with *roles* that had to be filled with persons, things, or places. For instance, the act of a murder has roles *murderer* and *murdered*, and these roles, which Lodwick called *appellative* nouns, name the individuals involved in the act within the context of the act, but outside this context the individuals have their own *proper* names, such as *man* or *beast* [56]. The grammatical function of predicates as carriers of action and their roles was later rediscovered by linguists like Bühler, Tesnière, and Fillmore [25]: Bühler noted that every word in a sentence has slots to be filled by others [10], and Tesnière presented the first dependency grammar [71], a grammar formalism in which the words of the language function as both terminal and non-terminal symbols [62]. As it

E-mail address: steimann@acm.org (F. Steimann).

turned out, dependency grammar is closely related to Sowa's conceptual structures [59,61,63], a knowledge representation formalism that has continuously gained recognition in the conceptual modelling community [22]. However, while the predicative or relational structures had already been embraced as central to modelling [14,17], the role concept, although equally fundamental, has long not received the widespread attention it deserved.

In recent years, interest in roles has grown continuously. But although there appears to be a general awareness that roles are an important modelling concept, until now no consensus has been reached as to how roles should be represented or integrated into the established modelling frameworks. This may partly be due to the different contexts in which roles are introduced, and partly to the different problems one is trying to solve with them. However, the divergence of definitions contradicts the evident generality and ubiquity of the role concept, and hampers its general acceptance as a modelling construct.

In this paper, we identify the basic properties of the role concept and combine them into a formal definition that fits in well with most modelling frameworks based on the classical dichotomy of types and relationships. For this purpose we review the characteristics that have been associated with roles, discuss their different realizations, and define a rudimentary modelling language named LODWICK that includes roles as a first class modelling concept. The formalization of roles in LODWICK differs from others in that it clarifies their relationship to specializations and generalizations, and in that it regards their natural role as intermediaries or bridges between relationships and the natural types populating their places.

The remainder of this paper is organized as follows. In Section 2, we give an informal account of what appear to be the natural properties of roles. In Section 3, we discuss how roles have been treated in the literature, roughly classifying the bulk of work into three major families: roles as named places of relationships, roles as specializations and/or generalizations, and roles as adjunct instances. The definition of LODWICK, its role concept and its evaluation against the properties identified in Section 2 follow in Section 4. That LODWICK, although only rudimentary, is a useful modelling language is shown by applying it to problems as diverse as the definition of design patterns, UML collaborations, object-oriented design and implementation, and metamodelling (Section 5). Section 6 completes the presentation by showing that LODWICK's role definition gives rise to a regard of polymorphism and substitutability that had formerly not been considered: objects playing roles can be considered polymorphic in the literal sense. We conclude with noting that LODWICK's definition of the role concept captures most of its semantics, while avoiding the disadvantages of its competitors.

2. What's in a role?

Bachman and Daya's *role data model* [4,5], an extension of the network model, is commonly credited as the first data model to have introduced an explicit notion of roles. [1] It was based on the observation that "most conventional file records and relational file *n*-tuples are role oriented. These files typically deal with employees, customers, patients, or students, all of which are role

[1] Although Falkenberg's *object-role model* [24] was published one year earlier, it appears that the roots of the role data model predate those of the object-role model.

types. This role orientation is in contrast with integrated database theory which has taught that each record should represent all aspects of some one entity in the real world. This difference in viewpoint has caused a great deal of confusion. The reason for the confusion is understood when it is realized that neither the roles of the real world nor the entities of the real world are a subset of the other" [4]. Unfortunately, the influence of the role data model on modelling has at best been modest even though it works out many of the concept's aspects that seem accepted today.

But the role data model was not only motivated by semantic considerations, it also solved a common practical problem. Because the network model allowed the members of a set to be records of various types, large pieces of redundant code had to be written in order to address the semantically equivalent, but syntactically distinct fields (items) of the members when iterating through a set. By letting all the entity types involved play the same role and by defining the fields as fields of the role, not of the types, it became possible to process all members with the same piece of code [5]. This feature of the role data model is immediately recognized as a form of polymorphism, another fundamental concept that pervades the literature on object-oriented modelling.

In his seminal monograph on *conceptual structures* Sowa distinguished between *natural types* "that relate to the essence of the entities" and *role types* "that depend on an accidental relationship to some other entity" [59, p. 82]. Although this distinction is quite clear and de facto introduces a new modelling concept, its potential effect on modelling practice is immediately weakened by letting both natural and role types coexist in the same type hierarchy, and by making no syntactic difference between the two besides, perhaps, calling one natural and the other role. In subsequent work of his, Sowa asserts that role types are subtypes of natural types. For example, the role types *Child*, *Pet*, *Quotient* and *Food* are subtypes of the natural types *Person*, *Animal*, *Number* and *Physical-Substance*, respectively [60]. This view is intuitively appealing and shared by many authors [7,23,36]. However, it also gives rise to a number of misconceptions, as will be seen.

In developing Sowa's ideas further, Guarino presented an ontological distinction between role and natural types [30]. A role, he maintained, is a concept that is founded and that lacks semantic rigidity, i.e., for a concept to be a role, it is required that its individuals stand in relation to other individuals, and that they can enter and leave the extent of the concept without losing their identity. A natural type, on the other hand, is characterized by semantic rigidity and lack of foundation: an individual of a natural type cannot drop this type without losing its identity, and for an individual to belong to that type it is not required that it stands in relationship to others. For example, *Person* is a natural type, because an individual, if a person, will always remain (and always has been) a person, and being a person is independent of the existence of any relationships. *Student*, on the other hand, is a role since to be a student enrollment in a university is required, and finishing studies does not lead to a loss of identity. Note that *Adolescent* and *Adult* are neither: lacking both semantic rigidity and foundation, they are *states* or *phases* of (the lifetime of) an individual. [2]

[2] Note that, to a certain extent, Guarino's criteria parallel Lodwick's definition of appelative and proper nouns: murderer and murdered are related to each other through a murder, and they lack semantic rigidity; they are roles. Man or beast, on the other hand, are rigid; they are natural types [65].

Most contemporary literature on roles in object-oriented and conceptual modelling is much more pragmatic. The following is a list of features that we have identified; note that some conflict with others, and hence that there is no single definition of roles integrating all of them.

1. *A role comes with its own properties and behaviour*. This basic property suggests that roles are types. And indeed, only few approaches do not regard roles as types; among these are the Entity–Relationship (ER) model and some of its relatives [14,17,24,33,34].

2. *Roles depend on relationships* [7,15,21,30,59]. As suggested by the work of Sowa and Guarino, a role is meaningful only in the context of a relationship. Although a fundamental characteristic, many definitions of the role concept do not consider it, so that the states or phases of an object are equally regarded as their roles.

3. *An object may play different roles simultaneously* [41,45,52,54,55,75,76]. This is one of the most broadly accepted properties of the role concept. Because a role is usually regarded as a type (item 1), it amounts to the multiple classification of objects.

4. *An object may play the same role several times, simultaneously* [41,52,55,75,76]. This is an equally fundamental finding, a frequent example of which being an employee holding several employments. Unlike with different roles, however, it does not correspond to multiple classification. The main reason to distinguish multiple occurrences in the same role is that each occurrence of the object in a role is associated with a different state (item 10). For example, an employee has one salary and one office address per job. (Not in [3,4,53].)

5. *An object may acquire and abandon roles dynamically* [3,29,41,43,49,55]. This is a dynamic property of the role concept that comes close to object migration or dynamic (re)classification [46,69]. However, the two are not necessarily the same; for example, Wieringa et al. [75] make an explicit distinction between dynamic classification and role playing.

6. *The sequence in which roles may be acquired and relinquished can be subject to restrictions* [52,69,75]. For example, a person can become a teaching assistant only after becoming a student. The usual sequence specifying formalisms are in use.

7. *Objects of unrelated types can play the same role* [4]. Although a fundamental observation complementing those of items 3 and 4, it is not acknowledged by all authors. From a theoretical point of view, this feature amounts to an alternative basis for inclusion polymorphism, as will be discussed in Section 6.

8 *Roles can play roles* [15,41,54,75,76]. This mirrors the condition that an employee (which is a role of a person) can be a project leader, which is then a role of the employee (but also another role of the person, although only indirectly). A rather technical subtlety that seems to require that roles are themselves instances.

9. *A role can be transferred from one object to another* [41,76]. It may be useful to let a concrete role dropped by one object be picked up by another, or even to specify the properties of a concrete role without naming a particular role player. For example, the salary of an open position may be specified independently of the person that will be employed.

10. *The state of an object can be role-specific* [41,52,75]. The state of an object may vary depending on the role in which it is being addressed. Together with item 4, i.e., the possibility of one object playing the same role multiply at the same time, this seems to suggest that each role played by an object should be viewed as a separate instance of the object.

11. *Features of an object can be role-specific* [3,29,43,52,55]. Attributes and behaviour of an object may be overloaded on a by-role basis, i.e., different roles may declare the same features, but

realize them differently. If an object plays several of these roles simultaneously, it responds according to the role in which it is being addressed.

12. *Roles restrict access* [29,41,52,76]. When addressed in a certain role, features of the object itself (or of other roles of the object) remain invisible. This corresponds to an object having different perspectives, facets, or aspects.

13. *Different roles may share structure and behaviour* [16,29,41,54]. This usually means that role definitions inherit from each other [29], but sometimes also that the definitions of roles rely on features of the objects playing them (delegation) [16,41].

14. *An object and its roles share identity* [3,29,41,55]. In the object-oriented world this entails that an object and its roles are the same, a condition that has been paraphrased as "a role is a mask that an object can wear" [8]. (Not in [75].)

15. *An object and its roles have different identities* [75]. This view, which is quite singular, is a tribute to the so-called *counting problem*. It refers to the situation in which instances counted in their roles yield a greater number than the same instances counted by the objects playing the roles. For example, the number of passengers taking a certain means of transportation in one week may be greater than the number of individual persons traveling with that means during the same period [75].

It should be clear from this list and what has been said above that the role concept is not some fancy modelling extra, but that it makes up for a lack in expressive power left by other modelling concepts. As one might expect there is not one ideal way of defining such a concept, but a number of competing approaches. Therefore, before presenting our own we take a quick tour through the major categories of role definitions that we have identified.

3. Three common ways of representing roles

Despite the many different properties associated with roles, the number of substantially differing definitions of the concept proposed is really quite small. In fact, all more or less adhere to one of three possible views: roles as named places of a relationship, roles as a form of generalization and/or specialization, and roles as separate instances joined to an object. [3]

3.1. Roles as named places

The simplest notion of a role is that of a named place in a relationship. Codd [17] noted that if two or more places of a relationship in his relational model were declared to be of the same type, then the name of this type would not suffice to distinguish these places. In such a case, role names should serve to identify the places in question. Today it is a common practice to give every place of a relationship a role name, which also serves as a column header in tabular printouts of the relationship's extent.

The ER model [14] and many of its extensions take up the practice of assigning role names to the entity types participating in relationships. In object-oriented analysis and design languages

[3] A similar distinction, namely between roles as specializations and roles as the places of a relationship, has been made by Reimer [53]; a brief discussion of five approaches representing roles as separate instances is given by Kappel et al. [39].

(including UML [48]) whose static models are based on the ER model, it has become common to drop relationship names altogether and use role names instead. For example, the *is-parent-of* (or *is-child-of*) relationship is sufficiently characterized by the role names *parent* and *child*. Note that this convention not only delivers suitable attribute names (after all, many relationships end up as pointer attributes), but also avoids the linguistic redundancy resulting from the circumstance that the predicative expressions chosen as relationship names frequently repeat the predicate's subject role (as in *is-parent-of*).

The most fundamental account of roles as named places is presumably that by Falkenberg [24]. In his *object-role model* (ORM) objects and roles are the sole primitives from which object types, associations (the equivalent of tuples) and association types are derived. As it turns out, ORM's data model, which allows nested associations (associations on associations), is very similar to the so-called *feature structures* heavily employed in the unification-based branches of computational linguistics and knowledge representation [6,13]; in fact, it is close to the linguistic roots of the role concept noted in the introduction. However, since it defines types exclusively in terms of their instances and the roles they play, it is purely extensional and misses the distinction between natural and role types.

Building on the object-role model *Nijssen's information analysis method* (NIAM) [33] and its descendants [34] provide a deeper account of the linguistic role of roles: they employ so-called fact types which are the direct equivalents of linguistic statements associating properties with and expressing the relationships of objects, as the sole data structure. Each fact type involves a number of roles which correspond to the places of a predicate. Hence, a role in NIAM is de facto a named place of a relationship.

By labeling the types involved in a relationship, defining roles as named places acknowledges that roles exist only in the context of relationships (item 2 above), and, to a certain extent, the dynamic and multiple classification of objects via the roles they play (items 3–5, and 7). However, it fails to account for the fact that roles come with their own properties and behaviour (item 1 and most of the rest of the list), a deficiency that is usually resolved by regarding roles as types in their own right (and not as mere labels of types).

3.2. Roles as specializations and/or generalizations

If roles are types, the question arises how the roles and the natural types of a model should be related. Seemingly, a role type is more specific than the natural type of its role players and its extent is smaller, which would make it a specialization (and hence a subtype). For instance, if *Person* is a natural type, then its roles *Customer* and *Supplier* would appear as its subtypes, as shown in Fig. 1(a).

Quite obviously, such a solution requires dynamic and multiple classification, since a person can change its roles and play several roles simultaneously. But this is not a real problem; *instance* or *object migration* is a well-investigated modelling concept that accounts for the dynamic change of classification associated with role playing ([46,69]). The real problem with viewing roles as subtypes is much subtler.

Naturally, not only persons, but also organizations like companies, etc., can be customers and suppliers. However, declaring *Customer* and *Supplier* as subtypes of both *Person* and *Organization* as depicted in Fig. 1(b) renders their extents subsets of the intersection of the two, which is either very small or empty, but in any case clearly not what is intended. Rather, *Customer* and *Supplier*

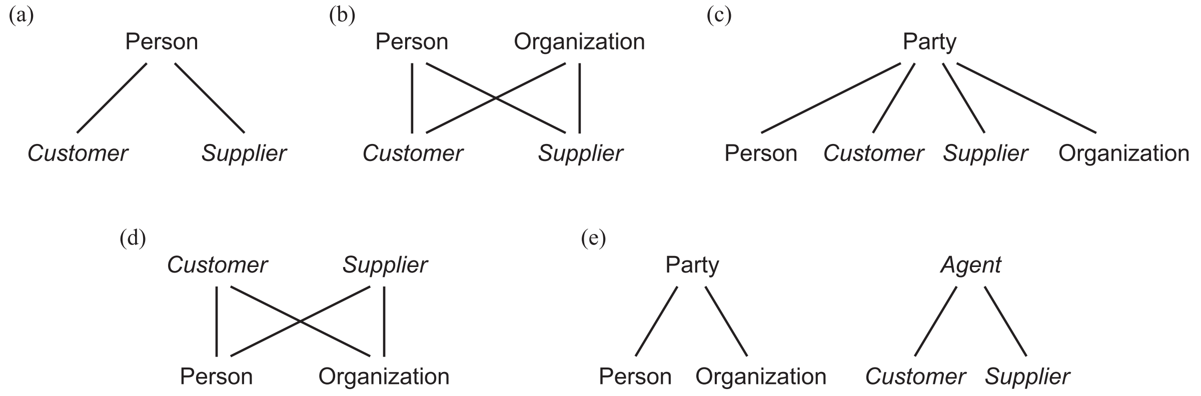

Fig. 1. Relating roles and types through subtyping (roles are italicized).

would subset the union of *Person* and *Organization, Party*, [4] but this leaves us with the question which of the subtypes of *Party* are natural and which are roles, and whether *Party* itself is a natural type or a role. In fact, Fig. 1(c) makes *Customer, Supplier, Person* and *Organization* all siblings, which is not only counterintuitive, but renders the type hierarchy heterogeneous: *Person* and *Organization* make a static partition of *Party* (every party is either a person or an organization), while *Customer* and *Supplier* are not even restrictions thereof (any party can also be a customer, a supplier, neither, or both).

The heart of the problem lies at the following misconception. Whereas dynamically (i.e., at any point in time) the set of customers and the set of suppliers are indeed each a subset of the set of parties, in principle every person and every organization can be a customer and a supplier or the instances of these types are not equal (and hence should not be instances of the same type). Statically, *Customer* and *Supplier* can therefore by no reasonable criterion be specified as restrictions of *Person, Organization*, or *Party*. In fact, it appears that viewing roles as subtypes is a consequence of an inadmissible intermingling of the dynamic nature of the role concept with the static properties of type hierarchies.

If roles are no subtypes, could they be supertypes? Surely, this contradicts the observation that roles are more specific than the types of their players. But this observation is false, anyway: a person, for example, has many properties not required of a customer or supplier – rather, being a customer or supplier imposes its required properties on persons and organizations, making the former supertypes of the latter. In fact, some authors treating roles as named places also acknowledge that more than one type can fill one place of a relationship – the *domains* of Kent [40] or the *multi-ET roles* of DB-Main [31] are unions or disjunctions of types that are declared for just that purpose. However, while regarding *Customer* and *Supplier* as supertypes of both *Person* and *Organization* (Fig. 1(d)) accounts for the fact that *all* persons and organizations *can* appear in these roles, it defies the dynamic viewpoint, namely that at any point in time only *some* of all persons and organizations existing at that time are customers and/or suppliers. This characteristic distinguishes roles from generalizations which,

[4] *Party* is a common catch-all type including the definition of *Person* as natural *and* legal entities [26].

including their subtypes both statically and dynamically, are supertypes; roles, on the other hand, are not. [5]

What we have is the paradoxical situation that, from the extensional point of view, roles are supertypes statically, while dynamically they are subtypes. And indeed, some authors maintain that specialization (the subtype relationship) and generalization (the supertype relationship) are not symmetric, in fact, that they are different relationships altogether [1,34,70], thus allowing the criticized heterogeneity of the type hierarchy of Fig. 1(c) to be eliminated. However, these authors base their distinction on the natural relatedness of the involved types, not on the different relationships of their extensions resulting from a difference in the static and the dynamic viewpoint. Consequently, they regard *Product* (which would be a role) as a generalization of *House* and *Car* [70], and *Student*, which is *the* prototypical role, as a specialization (in the usual sense) of *Person* [34]. Hence, this distinction does not add to the clarification of the relationship between roles and types.

The solution, clearly, lies in the separation of types and roles. If the type and role hierarchy are different hierarchies, none of the aforementioned problems related to subtyping occurs. For instance, in Fig. 1(e) every customer is also an agent, and so is every supplier, independently of whether the viewpoint is static or dynamic. What is missing, though, is a commitment to the relationship between natural and role types. Such a relationship has been introduced as the basis for defining roles as adjunct instances as treated in Section 3.3, and another one will be part of the formalization of roles in LODWICK (Section 4). First, however, we continue the discussion of a single hierarchy of types and roles.

By combining generalization, specialization, and the construction of domains for relationships whose places can be filled with instances of more than one type, Elmasri et al. [21] have suggested a category concept which is a subset of the union of a number of types. In the so-called *entity category relationship model* all relationships are not defined on entity types, but on categories, which seems to acknowledge the dependency of roles on relationships (item 2 above). However, categories are not uniquely linked to places – rather, the same category can occur more than once in the same relationship. Hence, categories are not roles in our sense.

Because of the temporal nature of the role concept, several authors investigate the possibility of roles as temporary specializations. A straightforward definition of roles as dynamic subtypes is that of Bock and Odell [7], which goes back to the *qua-classes* of KL-ONE [9] and is analogous to the *existence subclass* of SDM [35] and M.E.R.O.D.E. [58]. [6] According to this definition, a role is a type comprising all and only the objects that *currently* engage in a certain relationship (which is why it is also called a *current type*). For example, the relationship *marriage* between *Man* and *Woman* defines two role types, namely *Husband* and *Wife*, whose instances are all married. Marriage and divorce result in a dynamic reclassification of objects – the object migration phenomenon.

[5] Another criterion is that generalizations emphasize the common nature or *genus* of objects of different types, whereas roles emphasize their common use or function in a given context.

[6] The notion of role in KL-ONE [9] is not addressed here since it is merely another term for attribute, i.e., for a binary association as perceived from the defining concept's perspective. Note, however, that KL-ONE's attributes are much more elaborate than their object-oriented variants.

The interesting thing to note about the definition of Bock and Odell is that role types appear to be defined in terms of relationships or, rather, in terms of the dynamic extensions of relationships. On the other hand, the authors allow properties to be attached to role types just like they are attached to ordinary subtypes, so that the difference between role types and subtypes collapses to one being dynamically acquirable by instances, while the other is not. The acquisition of roles is made explicit by Papazoglou's approach [49,50] which allows that instances are directly assigned to and removed from the extent of a role-defining class so that the taking up of roles is always an explicit act and not a consequence of an instance participating in a relationship. As it turns out, this is the essence of most other definitions of roles as dynamic specializations [1,23,45,47].

Snoeck and Dedene also distinguish between static and dynamic specializations [58]. However, their notion of roles differs in that it extends the behaviour of a class and does not give rise to the definition of a new (sub)class. In particular, roles add new event types and sequences to an object type, but neither add nor refine attributes and methods. Consequently, unlike object types and their specializations roles cannot have instances of their own. This however lets roles appear as partial specifications of types [32] implemented as abstract supertypes in a multiple inheritance context. We shall come back to this later.

Finally, Jungclaus et al. [36,37] revert the view of roles as the dynamic variant of specialization by regarding specializations as special, namely static, kinds of roles. While this is a useful approach for their formalization of an object-oriented specification language integrating static and dynamic features of objects, like other accounts of roles as specializations it ignores some of the fundamental semantic differences between roles and (sub)types. In particular, as noted in [20], the fact that certain roles are shared by objects of different type suggests that the role concept should cover both dynamic specialization and generalization.

In general, representing roles as special kinds of specializations or generalizations amounts to the dynamic and multiple classification of objects. It also entails that an object and that object in its roles are represented by one instance, and this instance has only one state and only one identity. While this view harmonizes with most of the features of roles listed in Section 3 (namely, 1–3, 5–7, 11–14), it seems to preclude items 4, 8, and 15. The converse notion, namely that and object and its roles are different instances, promises to lift these restrictions; it will be considered next.

3.3. Roles as adjunct instances

Rather than struggling with the intricacies of modelling roles as generalizations or specializations and coping with the concomitant problem of multiple and dynamic classification, many authors prefer to realize roles as independent types the instances of which are carriers of role-specific state and behaviour, but not identity. An object and its roles are then related by a special *played-by* relation (or equivalent, with roles *role* and *role player*), and it is understood that an object and its roles form an aggregate (also referred to as *subject* [41]) that appears indivisible from the outside. The dynamic picking up of a role corresponds to the creation of a new instance of the corresponding role type and its integration into the compound, and dropping a role means releasing the role instance from the unit and destroying it.

This approach has several appealing implications. In particular, it appears as the natural realization of the characteristics 4, 5, 8, 10, and 12 listed in Section 3. In fact, since every instance has its own state, an object with roles automatically has many states, one per role it plays and one for the role-playing instance. Access restriction for the clients addressing an object in a role is but a natural side-effect, since the clients see only the object in the role, not its player or any other roles. For the same reason, the resolution of overloaded properties, i.e., properties that are defined for more than one role an object plays, is no problem with this approach. Note that access restriction and overloading does not prevent role instances from delegating requests to the objects they are affiliated to. In fact, in object-oriented design which has long struggled with the merits and pitfalls of inheritance, roles are being adopted as *the* modelling metaphor for delegation, the favoured alternative over subtyping [16].

Despite the uniform structure, the details from representing and object and the roles it plays vary from author to author. Reimer devised a framework of roles to parallel the usual subtyping and instantiation mechanisms of frame-based and object-oriented representation formalisms [53]. Two predefined relations are introduced: *may-be-a*, a second-order relation between classes and role classes (roughly corresponding to *is-a* or specialization), and *role*, conceived as a relation between an instance and a role class (corresponding to instantiation). The idea behind this doubling of concepts is that an instance of a class can also be a role instance of all role classes related to this class via the *may-be-a* relation. However, in order that this secondary role instantiation does not conflict with ordinary instantiation, role classes are instantiated as usual, and *role* relates the role player with role classes' instances (with these instances being the player's adjuncts). Therefore, although *role* is interpreted as a variant of instantiation, it is de facto a predefined first order relation that must either be heavily overloaded or declared on the most general of all role playing and role types.

Object specialization [57] is perhaps the most radical approach to modelling roles as separate instances. According to it, every object is represented by a hierarchy of instances. This hierarchy is an inheritance hierarchy: instances lower in the hierarchy inherit the properties of their ancestors (a rare case of instance-based inheritance). Each instance represents a role of the root object to whose hierarchy it belongs. Other objects have access to an object only via one of its roles which, due to the inheritance, represents the summary of all roles from the entry to the root of the hierarchy. Although this approach, convincing by its simplicity and expressiveness, has found its followers [38], the problem of split object identity is not addressed.

Several other approaches, although different in their basic definitions, arrive at similar results. Pernici [52] divides the specification of an object into several sections, each called a role. On object creation, a special role called base role plus any number of additional roles are instantiated. Furthermore, roles can dynamically be added and dropped as long as the specified role sequence specification is complied with. Aspects [55] builds on an implicit compatibility of types that is based on conformity (containment of declared signatures). An aspect of a (base) type is a separate type explicitly declared to be just that, and instantiation of an aspect always requires an instance of the corresponding base type to be provided. In Fibonacci, every object type is the root of a hierarchy of roles called the role family of the type. Roles can be acquired dynamically, but only one per role [3]. Because in Fibonacci roles specify only signatures and no implementation, the acquisition of a role requires the specification of an implementation so that different instances of the same type may behave differently in the same role.

Wieringa et al. [75] introduce role classes in contrast to dynamic subclasses. Every role class declares a role-player class via a (massively overloaded) *played-by* relation assigning an object or another role to each role instance. A role instance is an adjunct as in the other approaches, but with separate identity. This peculiarity is a tribute to the counting problem (cf. item 15 above) and is to be seen in contrast to the dynamic subclasses covering instance migration and dynamic classification. The ADOME system [43] is based on this approach and uses roles mainly to bridge between a statically typed database and the rules of knowledge base.

Kristensen and Østerbye [42] present an intuitive notation for the adornment of objects with roles. A role specifies and implements all properties that fall under one perspective of the object [41]. Roles can be bound to objects and to other roles. They can be transferred from one object to another, but they cannot exist on their own. The idea of transferable roles has independently been developed for DOOR [76], which regards roles as bridges between role owners and role players. The owner of a role is an object with an attribute of the role type. Properties of a role can be specified by the owner without an object (the role player) actually being assigned to the role. These properties are gained by an object once it assumes the role and lost once it drops it; the role is then vacant and ready to be played by another object.

Gottlob and coworkers [29] present an extension of SMALLTALK in which role hierarchies can be linked to the classes of the system's class hierarchy. Instances of a so linked class or of its subclasses can then dynamically assume and relinquish roles. A modification of SMALLTALK's test for identity allows role instances (as separate entities) to substitute for the objects they are a role of. Renouf and Henderson-Sellers [54] also provide methodical support for modelling roles of this kind and suggest an implementation pattern for EIFFEL.

A unifying conceptual approach that is independent of any implementation, but like ADOME [43] lends itself to extending existing implementations is the *entity-role-association* model by Chu and Zhang [15]. It divides the class hierarchy into a static part for the *is-a* classification of entity classes and a dynamic part for the *is-a* classification of role classes. The static and the dynamic part are connected via a *role-of* relation just like in the other approaches; however, associations (which are themselves classes) are exclusively defined on roles, a peculiarity that had already been suggested by Elmasri et al. [21] and that will be picked up in the definition of LODWICK in the next section.

Returning to Fig. 1(e), modelling roles as adjunct instances corresponds to declaring a *role-of* or *played-by* relationship between *Agent* and *Party*, which is then inherited to their subtypes. However, since this relationship relates instances, not types, it is first order and hence, even though it has predefined semantics, not on the same level as generalization and specialization. Also, because *Agent* and *Party* are not the only possible pair of role and role player, the *role-of* relationship must be overloaded for every other occurrence of this pattern. Nevertheless, the modelling of roles as adjuncts remains practically appealing; in fact, it has been recognized as the only legitimate object-oriented implementation of roles [16].

The biggest problem with viewing roles as adjunct instances is that it requires an unusual notion of instance, namely one according to which the instances of a role type do not have their own identity, but share it with others ("object schizophrenia"). This violates a basic assumption of object-orientation, namely that every object has its own identity, immutable and persistent, making it distinct from all others. This is not a technical necessity, but simply an appreciation of the understanding that every object of a model should correspond to a distinct, identifiable object of reality. In reality, however, the role of an object is not a different object, but merely its appearance in a given context. Hence the requirement that role instances should *not* have distinct identity.

Symptomatically, almost all work suggesting roles as adjunct instances has a strong implementational bias. This is presumably due to the authors' awareness of the concomitant conceptual problems mentioned above: modelling objects as collections of instances (a practice that is also referred to as *object slicing* [44] in object-oriented design and implementation), if not unsound, is unorthodox at least. Consequently the only theoretically ambitious approach regarding roles as adjuncts we are aware of is that of Wieringa et al. [75]; and this suggests that roles have separate identity.

4. Lodwick, a role-oriented modelling language

We now come to our own definition of the role concept. Since such a definition, a formal one especially, cannot exist in isolation, we embed it in a modelling language that is sufficiently developed to fix the semantics of roles, but that is open enough to allow its extension in different directions. We call this language Lodwick, because Lodwick was one of the earliest proponents of a notion of roles capturing their dependency on context and time.

4.1. Formal definition

For the specification of Lodwick, we borrow from the logic of feature types as described in [2,6], which is order-sorted. In order-sorted logic, the interpretation of unary predicates as assigning properties to objects and thus as defining types is complemented by a notion of sorts representing explicit hierarchical type information. Order-sorted logic is therefore an ideal basis for the specification of conceptual and object-oriented modelling languages, especially since it allows one to inherit its formal semantics.

However, like standard logic order-sorted logic is inherently atemporal. Since the definition of roles depends on a notion of time, Lodwick cannot be a purely static modelling language. Although order-sorted logic has been extended to cover model dynamics [36,75], the definition of the role concept does not depend on how the dynamic aspects of a model are specified, only on that they are. Therefore, it suffices for the specification of Lodwick that a greatly simplified view of a dynamic model is taken.

A *model specification* in Lodwick consists of a signature, a static model, and a dynamic model. The signature of a model comprises:

- a set of *natural type symbols*, N, called *types* for short;
- an *N-indexed* family of pairwise disjoint sets of *instance names*, $(I_n)_{n\in N}$, called *objects* for short;
- a *type hierarchy* $(N, \leqslant_{NN})$ defined by a partial order $\leqslant_{NN}$ on N;
- a set of *role type symbols*, R, called *roles* for short;
- a role hierarchy $(R, \leqslant_{RR})$ defined by a partial order $\leqslant_{RR}$ on R;
- a *role-filler relationship*, $<_{NR}$, relating the types in N to the roles in R; and
- a 2^R-indexed family of sets of relationship names $(P_w)_{w\in 2^R}$.

For $a \in I_n$, we write $a{:}n$ and call a an *object of n*. For convenience, we define I as the set of all objects by

$$I := \bigcup_{n\in N} I_n.$$

Although the set of objects can be infinite in principle, for most modelling purposes a rather small number of *prototypical instances* suffices. To a certain extent, these prototypical instances substitute for variables: they serve as placeholders for other objects of their types, and they act as coreference symbols expressing that objects in different places of a model are the same.

For $n' \leqslant_{NN} n$, we call n' a *subtype* of n and n a *supertype* of n'. Likewise, for $r' \leqslant_{RR} r$, we call r' a *subrole* of r and r a *superrole* of r'. For a pair $n <_{NR} r$, we say that *n fills r*. Informally, we also say that a type n fills role r if there is a supertype n' of n that fills a subrole r' of r. Note that unlike types, roles do not have associated sets of instances; as will be seen, they recruit their objects from the types they are filled by.

Last but not least, we require that the P_w are pairwise disjoint and that $P_w = \varnothing$ for, $|w| < 2$, i.e., all relationships are at least binary and overloading is not allowed. For a relationship $p \in P_{\{r_1,\dots,r_m\}}$, we write $p : r_1 \dots r_m$. For convenience, P is defined as the set of all relationships of a model, i.e.

$$P := \bigcup_{w \in 2^R} P_w.$$

The relationships are indexed over sets of roles because every role occurs only once in a relationship, and the order in which roles occur is insignificant. However, to be able to use standard set notation in certain situations, a total order on the set of roles is assumed, and each occurrence of a tupel or Cartesian product with roles at its places is assumed to obey that order, i.e., to have arranged its elements accordingly.

Given a signature as above, a *static model* in LODWICK is defined as comprising all instances of types and relationships that ever exist, independently of any dynamic or contemporaneity (such as cardinality) constraints. Thus, a static model comprises:

- for every $n \in N$ a static extension ext(n) including all objects of n and those of its subtypes, i.e.

$$\mathrm{ext}(n) = I_n \cup \bigcup_{n' <_{NN} n} \mathrm{ext}(n'),$$

so that $\mathrm{ext}(n') \subseteq \mathrm{ext}(n)$ for $n' \leqslant_{NN} n$;

- for every $r \in R$, a static extension ext(r) including all objects of the types filling r, i.e.

$$\mathrm{ext}(r) = \bigcup_{n <_{NR} r' \leqslant_{RR} r} \mathrm{ext}(n)$$

so that $\mathrm{ext}(n) \subseteq \mathrm{ext}(r)$ for $n <_{NR} r$ and $\mathrm{ext}(r') \subseteq \mathrm{ext}(r)$ for $r' \leqslant_{RR} r$; and

- for every $p : r_1 \dots r_m$ with $\{r_1, \dots, r_m\} \in 2^R$, a static extension ext(p) including tuples of objects of the roles declared for p, i.e.

$$\mathrm{ext}(p) \subseteq \mathrm{ext}(r_1) \times \cdots \times \mathrm{ext}(r_m).$$

We write

$$[r_1 \rightarrow a_1 : s_1, \dots, r_m \rightarrow a_m : s_m] : p$$

for an element of ext(p) and call this element an *association* of p. Although the notation is reminiscent of that of feature types [2,6], it stands here for an instance of a relationship (or predicate). It follows from the definition of ext(p) that all its elements must be well typed, i.e., $a_1 \in \mathrm{ext}(r_1), \ldots, a_m \in \mathrm{ext}(r_m)$. Note that roles serve a dual purpose in the specification of a relationship: they label the places, and they constrain the types of the objects participating in the relationship's associations. Also note that as far as the extensions of types and roles are concerned, in the static model there is no difference between $\leqslant_{NN}, <_{NR}$, and $\leqslant_{RR}$ other than that they are defined on different sets. This, however, will be different with the dynamic model.

A *dynamic model* in LODWICK specifies all possible sequences of sets of objects and associations between them. Each such sequence is called a possible *course* of the model, and each such set a *snapshot* of the model at an associated time. In a way, the static model may be viewed as the temporal projection of the dynamic model, i.e., as the union of all possible snapshots independent of any allowable sequence.

The dynamic model specifies, for every time t in every possible course c

- a dynamic extension $\mathrm{ext}_{c,t}(n)$ for every $n \in N$ with

$$\mathrm{ext}_{c,t}(n) = \bigcup_{n' \leqslant_{NN} n} \mathrm{ext}_{c,t}(n') \subseteq \mathrm{ext}(n)$$

comprising all and only those objects existing at that time in that course;

- a dynamic extension $\mathrm{ext}_{c,t}(p)$ for every $p : r_1 \ldots r_m$ with $\{r_1, \ldots, r_m\} \in 2^R$ with

$$\mathrm{ext}_{c,t}(p) \subseteq (\mathrm{ext}_{c,t}(r_1) \times \cdots \times \mathrm{ext}_{c,t}(r_m)) \cap \mathrm{ext}(p)$$

comprising all and only those relationship instances existing at that time in that course; and

- for every $r \in R$ a dynamic extension $\mathrm{ext}_{c,t}(r)$ with

$$\mathrm{ext}_{c,t}(r) = \{a \in I | \forall \{r', r_2, \ldots, r_m\} \in 2^R, r \leqslant_{RR} r' \forall p : r' r_2 \ldots r_m \exists a_2, \ldots, a_m \in I : \\ [r' \to a, r_2 \to a_2, \ldots, r_m \to a_m] : p \in \mathrm{ext}_{c,t}(p)\}$$

comprising all and only those objects that actually participate in that role (or any superrole thereof) in at least one relationship instance of every relationship with that role.

Note that while the static extensions of roles are defined solely in terms of the extensions of the types filling these roles, the definition of the dynamic extensions requires that the objects engage in relationships. Thus, while the static extensions of roles are always bigger than or equal to the ones of their filling types, the dynamic extensions can be smaller. Hence, $<_{NR}$ has clearly different properties than $\leqslant_{NN}$ and $\leqslant_{RR}$. Also note that dropping the indices c and t from the definition of the dynamic extensions of roles results in a definition equivalent to that of the static extensions if only in some course and at some time every object of every type filling a particular role occurs in that role in every relationship with that role. In other words: the specification of a relationship must not preclude any objects of the specified types from participating in that relationship. This however is only a natural condition, since otherwise the objects of the types would not be all equal (cf. Section 3.2).

Returning to the example of Fig. 1, the combined natural and role type hierarchy in question is specified in LODWICK by the following declarations:

$$\begin{aligned} Person &\leqslant_{NN} Party \\ Organization &\leqslant_{NN} Party \\ Party &<_{NR} Customer \\ Party &<_{NR} Supplier \\ Customer &\leqslant_{RR} Agent \\ Supplier &\leqslant_{RR} Agent \end{aligned}$$

Unlike with representing roles as adjunct instances (Section 3.3), the type and the role hierarchies of Fig. 1(e) are linked by two instances of $<_{NR}$, which is a second-order relationship like $\leqslant_{NN}$ and $\leqslant_{RR}$, relating types, not instances. Together with the above definitions the declarations imply that any person, like any organization, can in principle be a customer and a supplier, but that at any moment only those are that participate in a relationship with these roles declared at its places. For instance, with the relationship *deliver* declared as

$$deliver : Customer\ Supplier$$

and

$$[Customer \rightarrow a : Person,\ Supplier \rightarrow b : Organization] : deliver$$

in the extension of *deliver* (static or dynamic), it follows that a and b, elements of the extensions of the natural types *Person* and *Organization*, respectively, are also in the extensions of the roles *Customer* and *Supplier*.

Quite clearly, the extensional specification of a model, i.e., the listing of all possible snapshots together with the possible sequences thereof, is unfeasible. Therefore, the extensions of a model specified in LODWICK are complemented by a set of intensions added as follows:

- for every $n \in N$ an intension int(n), specifying the properties of the type (including attributes, behaviour, and the lifecycles of the objects);
- for every $p \in P$ an intension int(p) specifying conditions (such as the required role types and cardinalities) on the possible associations in the relationship's extension; and
- for every $r \in R$ an intension int(r).

The intensions of roles are divided into an absolute part, $\text{int}_{\text{abs}}(r)$, specifying properties just like the intensions of types, and a relative part, $\text{int}_{\text{rel}}(r)$. The relative part refers to the fact that the role occupies a place in the relationships declared with that role, and that for an object to belong to the extension of the role it must engage in these relationships (the definition of the dynamic extension of roles above).

The intension of a natural type is inherited to its subtypes so that $\text{int}(n') \Rightarrow \text{int}(n)$ for $n' \leqslant_{NN} n$. The same holds for roles: $\text{int}(r') \Rightarrow \text{int}(r)$ for $r' \leqslant_{RR} r$. However, while the absolute *and* the relative part are inherited down the $\leqslant_{RR}$ relation, the semantics of the role-filler relation $<_{NR}$ is different: only $\text{int}_{\text{abs}}(r)$ is inherited to the types filling this role so that $\text{int}(n) \Rightarrow \text{int}_{\text{abs}}(r)$ for $n <_{NR} r$ – because natural types are independent of the relationships their instances can engage in, the inheritance of $\text{int}_{\text{rel}}(r)$ stops at the role/type-transition. This, and the definition of the extension of roles given above allows it that statically roles are supertypes of the roles filling them, whereas dynamically their extensions are only subsets.

Clearly, Lodwick is only a rudimentary modelling language. In particular, while some specification of the possible sequences of snapshots is assumed, it is not clear how these sequences are specified or controlled – a signaling or messaging mechanisms enabling object interaction and behaviour specification is missing. However, such mechanisms and their languages exist [36–38,58] and are considered extensions to Lodwick, and because the choice of an appropriate formalism has no influence on the definition of roles in Lodwick, it will not be dealt with here.

4.2. Expressiveness

To evaluate the adequacy of Lodwick's role concept, it is checked against the properties of roles identified in Section 2.

1. *A role comes with its own properties and behaviour*: Yes. Roles are types, only that they cannot be instantiated. However, since the absolute properties of a role are inherited to the types filling them, they influence the properties of the instances playing them.

2 *Roles depend on relationships*: Yes. Roles occupy the places of relationships, and the relative part of a role's intension captures which relationships an object must participate in to be considered playing the role.

3. *An object may play different roles simultaneously*: Yes. An object may occur in as many different roles within the same or different associations as allowed by the relationships' specifications.

4. *An object may play the same role several times, simultaneously*: Yes. An object can occur in the same role within different associations of the same or different relationships, as allowed by the relationship specifications.

5. *An object may acquire and abandon roles dynamically*: Yes. Roles are assumed by an object as associations with that object are added, and relinquished as associations are removed from the dynamic extensions of relationships.

6. *The sequence in which roles may be acquired and relinquished can be subject to restrictions*: Possible. The specification of sequences (of extensions, cf. item 5) lies in the responsibility of the dynamic model.

7. *Objects of unrelated types can play the same role*: Yes. This is one of the cornerstones of Lodwick's role formalization; it follows from the definition of the role-filler relation linking the type and the role hierarchy.

8. *Roles can play roles*: No. This is not possible, since roles have no instances of their own.

9. *A role can be transferred from one object to another*: Possible. This however would require the introduction of variables, which would be an extension to Lodwick.

10. *The state of an object can be role-specific*: Partly. The associations an object participates in contribute to its state. These associations can be extended to capture the state that is associated with the object as playing the role. For example, the different salaries of a person in different employee roles may be included in the *employ* relationship.

11. *Features of an object can be role-specific*: Possible. Role are types and as such come with their own features. Role features are inherited to the types filling the roles, but a role-sensitive resolution mechanism (qualification) is needed if the same features are inherited from more than one role.

12. *Roles restrict access*: Not applicable. Lodwick does not have notions of accessibility or visibility.

13. *Different roles may share structure and behaviour*: Partly. As noted under item 11, the features of role specifications are inherited down the role hierarchy to the types filling the roles. Vice versa, properties of the types filling roles are not inherited to these roles. For instance, if the type *Person* has a *placeOfBirth* attribute, this attribute is not shared by its role *Customer*. This however makes sense since not all customers are persons.

14. *An object and its roles share identity*: Yes. An object in a role is the object itself.

15. *An object and its roles have different identities*: No. This follows from item 14.

Overall, it appears that LODWICK covers most of the features expected from roles. This may seem a little surprising, since many of the items appeared to be bound to the representation of roles as adjunct instances, which LODWICK denies. Instead, an object in a role is modelled as an object participating in a relationship in the place of that role, and as many different engagements as feasible are allowed. The general ability of objects to play a certain role is specified by a declaration listing all types filling that role, and by a role hierarchy that is independent of the type hierarchy. In principle, LODWICK's roles are no more dynamic than its types, but just as instances of a type can come into existence and cease to exist by adding and removing them from the dynamic extension of their types, they can assume and drop roles by participating in associations being added and removed from the dynamic extensions of their relationships.

5. Applications and extensions

LODWICK as a modelling language is quite simple. This is deliberate, since one of the objectives of our work is to show that roles are a modelling primitive and not some add-on bell or whistle. To give evidence to this, we go through a number of examples, indicating possible extensions of the language where in place.

5.1. Pattern specifications

Design patterns [27] are a common way of specifying recurring pieces of object-oriented designs and implementations. Design patterns are often specified as class diagrams enhanced with interaction specification where necessary. However, it has been noted that not classes, but roles are the true structural primitives upon which most patterns rely [11], and that classes are only used because a role modelling primitive is not yet established.

The composite pattern is a frequent pattern underlying parts explosions, directories, and analogous recursive structures. A composite consists of components, which are either atomic or themselves composites [27]. In LODWICK, this is expressed by a relationship *consist* with two corresponding roles:

$$consist : Composite\ Component.$$

Note that this declaration is completely independent of what is being composed – the pattern is instantiated only by declaring the role-filling types. For a directory structure comprised of folders and files, the necessary declarations are

$$Folder <_{NR} Composite \quad File <_{NR} Component \quad Folder <_{NR} Component.$$

Accordingly, for a parts explosion the role-filling types are *PartsList* and *Part*; by means of corresponding declarations, their instances can populate the same relationship.

However, because the declaration of *Folder*/*File* and *PartsList*/*Part* as role fillers of *Composite*/*Component*, respectively, is not coordinated, associations of the kind

$$[\mathit{Composite} \to p : \mathit{PartsList},\ \mathit{Component} \to f : \mathit{File}] : \mathit{consist}$$

are well typed and thus admissible. This is a common problem of disjunctions and requires some kind of role coordination in the specification (the intension) of the relationship. A possible solution is to overload the declaration of *consist* with subroles of *Composite* and *Component*

$$\mathit{consist} : \mathit{PartsListComposite}\ \mathit{PartsListComponent}$$

and

$$\mathit{consist} : \mathit{DirectoryComposite}\ \mathit{DirectoryComponent},$$

respectively, and overloading is a possible extension of LODWICK.

5.2. Roles in UML

In the UNIFIED MODELING LANGUAGE (UML), roles serve two purposes: they label association ends (the UML term for places of relationships), and they act as type specifiers in the scope of a collaboration (so-called classifier roles; the association and association end roles of UML are not considered here) [48]. Because LODWICK does not distinguish between the two purposes, an inevitable clash occurs when trying to specify a UML-collaboration in LODWICK, because the association ends' rolenames and the classifier roles do not match. An example of this is shown in Fig. 2.

LODWICK's solution to this problem is simple. Since the roles *Teacher* and *Student* occur in the places of several relationships all of which have their own roles assigned, it is assumed that *Teacher* is a subrole of *faculty member*, *lecturer*, and *tutor*, whereas *Student* is a subrole of *student* and *participant*. In fact, it seems reasonable to assume that the role of a teacher unifies the roles of a faculty member, a lecturer and a tutor, because being a teacher requires the properties of all of them. Note that for the classifier roles filled by the classes (natural types) *Faculty* and *Course* no clash with the rolenames occurs since they are unnamed; nevertheless, *taken course* and *given course* are different roles filled by *Course*, which is either resolved by $\mathit{Course} <_{NR} \mathit{given\ course}$ and $\mathit{Course} <_{NR} \mathit{taken\ course}$, or by introducing a common subrole for the two that is filled by *Course* [67]. The same applies to *Faculty*.

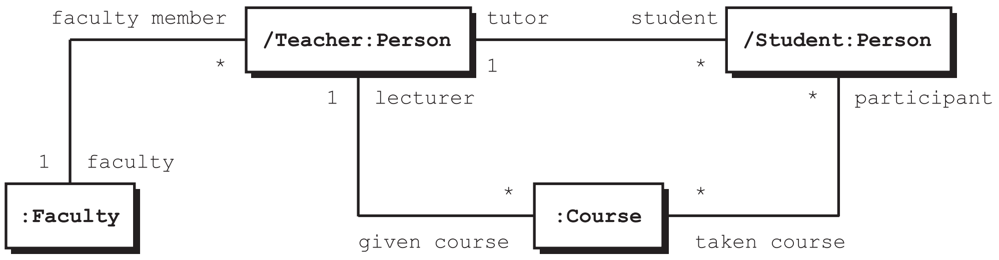

Fig. 2. A typical UML collaboration with classifier roles shown as boxes (taken from [48]). The name after the slash is the name of the classifier role, and the name after the colon is that of the type (class) on which the role is based. Note that :*Faculty* and :*Course* provide no explicit role names. (See [67] for a complete discussion of roles in UML.)

5.3. Object-oriented design and implementation

LODWICK is intended to be an exploratory language for object-oriented modelling with roles at the conceptual level. However, conceptual or object-oriented modelling is usually only a precursor to design and implementation, and if the role concept of LODWICK is to be of any practical value, it has to find its way into software reality.

Most mainstream object-oriented programming languages rely on classes as the primary structuring construct. However, classes can be abstract, i.e., not instantiable, and as such they can be used to specify interfaces. Interfaces comprise protocols of behaviour and specifications of other features that are relevant for the use of a class. Since a class can implement several interfaces, each interface is only a partial specification of the properties associated with a class. [7]

Now the type and role hierarchy of a model specification in LODWICK can be mapped to the class hierarchy of an object-oriented program as follows [66]:

- every type is mapped to a class and each of its subtypes to a subclass of that class;
- every role is mapped to an abstract class (interface) and each of its subroles to an abstract subclass (sub-interface) of that class (interface); and
- every role-filler relationship between a type and a role is mapped to a subclass relationship between the corresponding classes (an implements relationship between the corresponding class and interface).

Note how this convention accounts for the inclusion of the static extensions of types and roles as specified for LODWICK in Section 4.1, and for the inheritance of properties from roles to their subroles to the types filling the roles. However, it cannot account for the differences in the dynamic extensions and their connection to the extension of relationships. This is in the responsibility of another convention.

Since the places of the relationships of a model usually end up as the variables (instance variables and formal parameters) of an object-oriented program, LODWICK's definition of the role concept and its mapping to abstract classes or interfaces suggests that all variables should declare abstract classes or interfaces as their types. The dynamic extension of a role could then be defined as the set of all objects assigned to variables of the corresponding type. Even though this does not match with the formal definition of the dynamic extension of roles given in Section 4.1, it results in a rather intuitive notion of roles in object-oriented programs. Besides, declaring variables with abstract types is considered good programming practice anyway [18,27,66], and LODWICK as a modelling language is only promoting this practice.

5.4. Metamodelling

LODWICK is defined after the common object language/metalanguage pattern. The metalanguage has notions of sets, relations, etc., which are assumed to be pre-existing concepts. The elements of LODWICK as an object language have been defined as needed, but exclusively in terms of the metalanguage. As it turns out, modelling languages (as object languages) often have the same elements as their metalanguages. This is not only a source of continuous confusion, but also gives rise to the conception that the metalanguage is itself the object language of a higher ranking

[7] This of course requires multiple inheritance.

meta-metalanguage, and so forth. This would obviously result in an infinite recursion if not at some level certain ontological commitments were made, namely as to what is actually assumed as pre-existing. Since such a commitment almost inevitably includes sets and relations, there seems not much point in thinking about a metamodel for LODWICK.

Except for one thing. We have contended that roles, like objects and relations, are so fundamental that they should be an integral part of any modelling language, and this includes the metalanguage. And indeed, $\leqslant_{NN}$ for example is a metalevel relationship with two roles, namely *subtype* and *supertype*, both being filled by the set of types N. The same holds for $\leqslant_{RR}$ with roles *subrole* and *superrole* and R as the role-filling type. Likewise, LODWICK's $<_{NR}$ is a metalevel relationship with roles *filler* and *filled* such that $N <_{NR}$ *filler* and $R <_{NR}$ *filled.*

The fact that the elements of an object language are easily transferred to the metalanguage is usually a good indication that the elements are well-chosen. Trying to do this with roles as adjunct instances, however, does not lead to satisfactory results: either we deny the existence of roles at the metalevel, or we have to model a role like *subtype* as an adjunct to *type*, which is hardly natural. Note that a similar problem arises from arguing that the representation of roles as adjunct instances is not a modelling primitive, but a design pattern in character, because the fact that design pattern relate roles, not classes (cf. above), raises the question of how the roles of the role pattern, *namely role* and *role player*, are to be represented.

6. Roles and polymorphism

Literally, polymorphic is taken to mean "having, assuming or passing through many or various forms, stages, or the like" [72]. Typical manifestations of polymorphism are the insects of the order Lepidoptera with forms caterpillar and butterfly, and carbon with forms coal, graphite, and diamond. It appears that in this original sense, polymorphism is a property of an individual object stating that this object has characteristics – simultaneously or in sequence – so different that they would normally be attributed to different objects. However, in the computing community, polymorphic usually denotes a property of an operator or function symbol, namely that its meaning [28] or associated behaviour [51] is determined by its operands or parameters (rather than the symbol alone).

Strachey, who introduced the term to the computing field [68], made a distinction between what he called *ad hoc polymorphism* and *parametric polymorphism*. According to his definition, ad hoc polymorphism denotes the random reuse of symbols (such as letting + denote number addition, string concatenation, and Boolean disjunction); it is further divided into *overloading* and *coercion*. Parametric polymorphism refers to the construction of types from a template and a number of parameters (as in *List*(*T*), with instantiations *List*(*Number*), *List*(*String*), etc.). Cardelli and Wegner complement parametric polymorphism with what they call *inclusion* or *inheritance polymorphism*, which basically refers to substitutability as a consequence of the inclusion of types in a subsumption hierarchy [12,73]. Both are special forms of what they call *universal polymorphism*, the kind of polymorphism that allows polymorphic functions to operate on an unlimited number of types. This is to be seen in contrast with ad hoc polymorphism, whose functions will work only on a finite (and potentially unrelated) set of types, namely on the types the overloading or coercion are explicitly declared for.

The polymorphism of functions and operators is a particularly appealing concept when a family of related types is in use. For example, if the usual arithmetic operators are defined on the family of number types (a type hierarchy including *Real*, *Rational*, *Integer*, and *Natural* in descending order), each operator must make provisions to cope with the different combinations of its operands' types. The polymorphism of operators is pushed even further if the result or return type of an operator is not only determined by the types of its operands, but also by their values. For instance, the subtraction of two natural numbers can result in an integer, or in a natural number. Last but not least, instance creation may be polymorphic, too: depending on the parameters supplied, the type of the new instance may vary [64].

All this has little to do with the encyclopedic definition of polymorphism given above: rather than *same* objects having *different* forms, objects of *different* types implement the *same* functions. What one would expect instead is that polymorphism is a property of single types or, rather, their instances. Only inclusion polymorphism addresses this point to a certain extent: if the set of instances of a given type includes the instances of its subtypes, then these instances differ in form so that the type could indeed be called polymorphic. However, that an instance of a type is also an instance of its supertypes is not a sufficient condition for its being polymorphic, since supertypes are generally not manifestations of different form, but of different levels of abstraction. After all, being a mammal and being a vertebrate are not different forms of a person, but different (biological) abstractions. In fact, the form imposed on the instances by their type always entails the forms imposed by their supertypes. Thus, abstraction or generalization hierarchies do not make instances polymorphic.

This is where roles come into play. Being a student, an employee and/or a mother are different "forms" of a person, each with different characteristics, each specified by its own role type. An instance is polymorphic if it can play different roles, and the different forms of the instance are defined by the roles it can play. This corresponds to the encyclopedic definition of polymorphism above: an object in different roles has characteristics so different that they would normally be ascribed to different objects (the classical fallacy noted by Bachman and Daya [4]); yet it is the same object.

On the other hand, LODWICK's definition of roles also accounts for polymorphism à la Cardelli and Wegner, since different types of objects may play the same role and thus implement the same functions. This includes inclusion polymorphism (in case the types are all subtypes of a common supertype defining the functions) as well as ad hoc polymorphism (in case the types are unrelated so that the implementation is not inherited). Thus, LODWICK's role-based polymorphism is more general: it accounts for both same objects having different forms (the encyclopedic definition) and for different objects having same form (the computational definition).

Last but not least, LODWICK's role concept lends itself to a redefinition of the notion of substitutability, which is closely related to the definition of inclusion polymorphism [74]. In fact, if every object that can play a given role guarantees to fulfill whatever the role requires of its role players, then we have the following *modified principle of substitutability*: an instance of a type can always be used in any context in which an instance playing a role filled by that type is expected. In other words: instances of different types can substitute for each other if the types fill the same role and if the instance required is an instance playing that role – the very definition of plugability [16,18].

7. Conclusion

All definitions of roles discussed here have their merits and drawbacks. For example, viewing a role as a named place of a relationship may appear trivial but stresses that roles exist only in context, a fact that is only too often neglected by other definitions. Roles are no subtypes, but viewing them as supertypes poses problems, too. Last but not least, representing roles as adjunct instances provides a useful metaphor for modelling delegation, yet it is conceptually questionable. In essence, it appears that the role concept is a truly original one, one that cannot be emulated by any of the better established conceptual or object-oriented modelling constructs. The challenge of defining a suitable role concept is to integrate it into existing modelling frameworks causing as little redefinition as necessary, while capturing as much of its semantics as possible. By introducing a formal definition of roles that clarifies their status with respect to generalization and specialization, and that places roles as intermediaries between relationships and the natural types populating their places, we believe that we have succeeded in both respects.

References

[1] S. Abiteboul, R. Hull, IFO: A formal semantic database model, ACM Transactions on Database Systems 12 (4) (1987) 525–565.
[2] H. Aït-Kaci, R. Nasr, LOGIN: A logic programing language with built-in inheritance, Journal of Logic Programming 3 (1986) 185–215.
[3] A. Albano, R. Bergamini, G. Ghelli, R. Orsini, An object data model with roles, in: R. Agrawal, S. Baker, D. Bell (Eds.), Proceedings of the 19th International Conference on Very Large Databases, Morgan Kaufmann, Dublin; 1993, pp. 39–51.
[4] C.W. Bachman, M. Daya, The role concept in data models, in: Proceedings of the Third International Conference on Very Large Databases, 1977, pp. 464–476.
[5] C.W. Bachman, The role data model approach to data structures, in: S.M. Deen, P. Hammersley (Eds.), Proceedings of the International Conference on DataBases, University of Aberdeen, Heyden & Son, 1980, pp. 1–18.
[6] K.H. Bläsius, U. Hedtstück, C.R. Rollinger (Eds.), Sorts and types in artificial intelligence, Lecture Notes in Artificial Intelligence 418, Springer, Berlin, 1989.
[7] C. Bock, J.J. Odell, A more complete model of relations and their implementation: Roles, Journal of Object-Oriented Programming 11 (2) (1998) 51–54.
[8] G. Booch, Object-Oriented Analysis and Design with Applications, Addison-Wesley, Menlo Park; 1994.
[9] R. Brachman, J. Schmolze, An overview of the KL-One knowledge representation scheme, Cognitive Science 9 (2) (1985) 171–216.
[10] K. Bühler, Sprachtheorie, Fischer, 1934.
[11] F. Buschmann, Falsche Annahmen (Teil 2), OBJEKTspektrum 4 (1998) 84–85.
[12] L. Cardelli, P. Wegner, On understanding types data abstracting and polymorphism, ACM Computing Surveys 17 (4) (1985) 471–522.
[13] B. Carpenter, The Logic of Typed Feature Structures: With Applications to Unification Grammars Logic, Programs and Constraint Resolution, Cambridge University Press, Cambridge; 1992.
[14] P.P. Chen, The entity-relationship model: Towards a unified view of data, ACM Transactions on Database Systems 1 (1) (1976) 9–36.
[15] W.W. Chu, G. Zhang, Associations and roles in object-oriented modeling, in: D.W. Embley, R.C. Goldstein (Eds.), Proceedings of the 16th International Conference on Conceptual Modeling: ER'97, Springer, Berlin; 1997, pp. 257–270.
[16] P. Coad, M. Mayfield, JAVA Design: Building Better Apps and Applets 2. Ausgabe, Yourdon Press, Upper Saddle River; 1999.
[17] E.F. Codd, A relational model of data for large shared data banks, Communications of the ACM 13 (6) (1970) 377–387.
[18] D.F. D'Souza, A.C. Wills, Objects, Components and Frameworks with UML, Addison-Wesley, Reading, MA; 1998.
[19] U. Eco, La ricerca della lingua perfetta nella cultura europea, Laterza, Roma; 1993.
[20] H.D. Ehrich, R. Jungclaus, G. Denker, Object roles and phases, in: U.W. Lipeck, G. Koschorreck (Eds.), Proceedings of the International Workshop on Information Systems – Correctness and Reusability (IS-CORE '93) Informatik-Berichte 1/93 Universität Hannover, Institut für Informatik, Hannover, 1993, pp. 114–121.
[21] R. Elmasri, J. Weeldreyer, A. Hevner, The category concept: An extension to the entity relationship model, Data & Knowledge Engineering 1 (1) (1985) 75–116.

[22] ER Conference Series, Springer, Germany.
[23] L.J.B. Essink, W.J. Erhart, Object modelling and system dynamics in the conceptualization stages of information systems development, in: F. van Assche, B. Moulin, C. Rolland (Eds.), Proceedings of the IFIP TC8/WG8.1. Working Conference on the Object Oriented Approach in Information Systems, North-Holland, Amsterdam; 1991, pp. 89–116.
[24] E. Falkenberg, Concepts for modelling information, in: G.M. Nijssen (Ed.), Proceedings of the IFIP Conference on Modelling in Data Base Management Systems, North-Holland, Amsterdam; 1976, pp. 95–109.
[25] C.J. Fillmore, Types of lexical information, Auszüge abgedruckt, in: R. Dirven, G. Radden (Eds.), Fillmore's Case Grammar: A Reader, Julius Groos Verlag, Heidelberg; 1987.
[26] M. Fowler, Analysis Patterns: Reusable Object Models, Addison-Wesley, Menlo Park; 1997.
[27] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, New York; 1995.
[28] J.A. Goguen, J. Meseguer, Order-sorted Algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations, Theoretical Computer Science 105 (2) (1992) 217–273.
[29] G. Gottlob, M. Schrefl, B. Röck, Extending object-oriented systems with roles, ACM Transactions on Information Systems 14 (3) (1996) 268–296.
[30] N. Guarino, Concepts, attributes and arbitrary relations, Data & Knowledge Engineering 8 (1992) 249–261.
[31] J.L. Hainaut, Specification preservation in schema transformations: Application to semantics and statistics, Data & Knowledge Engineering 19 (2) (1996) 99–134.
[32] D.C. Halbert, P.D. O'Brien, Using types and inheritance in object-oriented programming, IEEE Software 4 (5) (1987) 71–79.
[33] T.A. Halpin, Conceptual Schema and Relational Database Design, Prentice-Hall, Sidney; 1995.
[34] T.A. Halpin, H.A. Proper, Subtyping and polymorphism in object-role modelling, Data & Knowledge Engineering 15 (1995) 251–281.
[35] M. Hammer, D. McLeod, Databse description with SDM: A semantic database model, ACM Transactions on Database Systems 6 (3) (1981) 351–386.
[36] R. Jungclaus, G. Saake, T. Hartmann, C. Sernadas, Object-Oriented Specification of Information Systems: The TROLL Language Informatik Berichte 91-04 TU Braunschweig, Braunschweig, 1991.
[37] R. Jungclaus, Modeling of Dynamic Object Systems – A Logic-Based Approach Dissertation, Verlag Vieweg, Wiesbaden; 1993.
[38] G. Kappel, M. Schrefl, Object/behaviour diagrams, Proceedings of the Seventh International Conference on Data Engineering, IEEE Computer Society Press, Los Alamitos; 1991, pp. 530–539.
[39] G. Kappel, W. Retschitzegger, W. Schwinger, A comparison of role mechanisms in object-oriented modeling, in: K. Pohl, A. Schärr, G. Vossen (Eds.), Modellierung '98 Bericht Nr. 6/98-I, Angewandte Mathematik und Informatik, Universität Münster, 1998, pp. 105–109.
[40] W. Kent, Data and Reality: Basic Assumptions in Data Processing Reconsidered, North-Holland, Amsterdam; 1978.
[41] B.B. Kristensen, Object-oriented modeling with roles, in: J. Murphy, B. Stone (Eds.), OOIS '95: Proceedings of the International Conference on Object-Oriented Information Systems, 18–20 December 1995, Dublin, Springer, 1996, pp. 57–71.
[42] B.B. Kristensen, K. Østerbye, Roles: Conceptual abstraction theory and practical language issues, Theory and Practice of Object Systems 2 (3) (1996) 143–160.
[43] Q. Li, F.H. Lochovsky, ADOME: An advanced object modeling environment, IEEE Transactions on Knowledge and Data Engineering 10 (2) (1998) 255–276.
[44] J. Martin, J.J. Odell, Object-Oriented Analysis and Design, Prentice-Hall, Englewood Cliffs; 1992.
[45] G. Maughan, B. Durnota, MON: An object relationship model incorporating roles, classification, publicity and assertions, in: Proceedings of OOIS '94, International Conference on Object Oriented Information Systems, 1993.
[46] A.O. Mendelzon, T. Milo, E. Waller, Object migration, in: Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems PODS, 1994, pp. 232–242.
[47] M.C. Norrie, A. Steiner, A. Würgler, M. Wunderli, B. Thalheim, A model for classification structures with evolution control, 15th International Conference on Conceptual Modeling Proceedings: ER '96, Springer, Berlin; 1996, pp. 456–471.
[48] OMG, Unified Modeling Language Specification V 1.3, June 1999 (http://www.omg.org).
[49] M.P. Papazoglou, Roles: A methodology for representing multifaceted objects, in: D. Karagiannis (Ed.), Proceedings of the International Conference on Database and Expert Systems Applications (DEXA 91), Springer, Berlin; 1991, pp. 7–12.
[50] M.P. Papazoglou, Unraveling the semantics of conceptual schemas, Communications of the ACM 38 (9) (1995) 80–94.
[51] D.M. Papurt, Generalization and polymorphism, Report on Object Analysis and Design 2 (5) (1996) 13–16.
[52] B. Pernici, Objects with roles, in: F.H. Lochovsky, R.B. Allen (Eds.), Proceedings of the Conference on Office Information Systems, SIGOIS Bulletin, vol. 11, no. 2/3, ACM Press, New York, 1990, pp. 205–215.
[53] U. Reimer, A representation construct for roles, Data & Knowledge Engineering 1 (3) (1985) 233–251.
[54] D.W. Renouf, B. Henderson-Sellers, Incorporating roles into MOSES, in: C. Mingins, B. Meyer (Eds.), Proceedings of the 15th International Conference on Technology of Object-Oriented Languages and Systems: Tools 15, Prentice Hall, New York; 1995, pp. 71–82.

[55] J. Richardson, P. Schwartz, Aspects: Extending objects to support multiple, independent roles, in: J. Clifford, R. King (Eds.), Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, SIGMOD Record ACM Press, vol. 20, no. 2, 1991, pp. 298–307.
[56] V. Salmon, The Works of Francis Lodwick, Longman, London; 1972.
[57] E. Sciore, Object specialization, ACM Transactions on Infomation Systems 7 (2) (1989) 103–122.
[58] M. Snoeck, G. Dedene, Generalization/specialization and role in object oriented conceptual modeling, Data & Knowledge Engineering 19 (2) (1996) 171–195.
[59] J.F. Sowa, Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, New York; 1984.
[60] J.F. Sowa, Using a lexicon of canonical graphs in a semantic interpreter, in: M.W. Evens (Ed.), Relational Models of the Lexicon: Representing Knowledge in Semantic Networks, Cambridge University Press, Cambridge; 1988, pp. 113–137.
[61] J.F. Sowa, Semantic networks, in: S.C. Shapiro (Ed.), Encyclopedia of Artificial Intelligence, second ed., Wiley, New York, 1992, pp. 1493–1511.
[62] F. Steimann, C. Brzoska, Dependency unification grammar for prolog, Computational Linguistics 21 (1) (1995) 95–102.
[63] F. Steimann, Dependency parsing for medical language and concept representation, Artificial Intelligence in Medicine (1998) 77–86.
[64] F. Steimann, The family pattern, Journal of Object-Oriented Programming, in press.
[65] F. Steimann, Eine Grundlegung des Rollenbegriffs für die objektorientierte Modellierung, mit dem Vorschlag einer Änderung von UML, in: J. Ebert, U. Frank (Hrsg) Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik: Modellierung, Fölbach-Verlag, Koblenz, 2000, pp. 55–69.
[66] F. Steimann, Role = Interface: A merger of concepts, Journal of Object-Oriented Programming, 2001, to appear.
[67] F. Steimann, A radical revision of UML's role concept, Technical Report, www.kbs.uni-hannover.de/~steimann, 2000.
[68] C. Strachey, Fundamental concepts in programming languages, in: Lecture Notes for International Summer School for Computer Programming, Copenhagen, 1967.
[69] J. Su, Dynamic constraints and object migration, in: G.M. Lohman, A. Sernadas, R. Camps (Eds.), Proceedings of the 17th International Conference on Very Large Databases, VLDB Endowment Press, Sarattoga; 1991, pp. 233–242.
[70] A.H.M. Ter Hofstede, Th.P. van der Weide, Expressiveness in conceptual data modelling, Data & Knowledge Engineering 10 (1) (1993) 65–100.
[71] L. Tesnière, Éléments de syntaxe structurale, second ed., Librairie C Klincksieck, Paris; 1965.
[72] Webster's Encyclopedic Unabridged Dictionary of the English Language, Random House, New York, 1996.
[73] P. Wegner, The object-oriented classification paradigm, in: P. Shriver, P. Wegner (Eds.), Research Directions in Object-Oriented Programming, MIT Press, Cambridge, MA; 1987, pp. 479–560.
[74] P. Wegner, B. Zdonik, Inheritance as an incremental modification mechanism or what like is and isn't like, in: S. Gjessing, K. Nygaard (Eds.), ECOOP '88: European Conference on Object-Oriented Programming, Proceedings, Springer, Berlin, 1988, pp. 55–77.
[75] R. Wieringa, W. de Jonge, P. Spruit, Using dynamic classes and role classes to model object migration, Theory and Practice of Object Systems 1 (1) (1995) 61–83.
[76] R.K. Wong, H.L. Chau, F.H. Lochovsky, Dynamic knowledge representation in DOOR, in: X. Wu, J. Tsai, N. Pissinou, K. Makki (Eds.), Proceedings of the 1997 IEEE Knowledge and Data Engineering Exchange Workshop, IEEE Computer Society, Los Alamitos; 1997, pp. 89–96.

Friedrich Steimann graduated in Informatics at the Universität Karlsruhe. After working as a research engineer in industry (Alcatel Austria Elin Research Center), he went back to university (Technische Universität Wien), where he obtained his Doctor technicae for work on the diagnostic monitoring of clinical time series. Following short interludes in Hildesheim and Braunschweig, he is now with the Universität Hannover, finishing his habilitation. Friedrich Steimann is currently working on the application of fuzzy sets in medical AI, trend detection and diagnosis in dynamic systems, and object-oriented modelling. He is a member of the ACM and of the Advisory Board of Artifcial Intelligence in Medicine (Elsevier).