Applied Ontology 2 (2007) 89–103
IOS Press

The role data model revisited

Friedrich Steimann

Lehrgebiet Programmiersysteme, Fernuniversität in Hagen, D-58084 Hagen, Germany
E-mail: steimann@acm.org

Abstract. While Bachman's role data model is often cited, it appears that its contribution, the introduction of role types and their conception as unions of entity types that occupy the places of relationship types, has mostly been ignored. This is unfortunate since it has led to countless reinventions of the wheel, and sometimes even to regress. With this homage, the author wishes to shed some light on the natural elegance of Bachman's role concept, and to make clear why he believes that it is valid until this day.

Keywords: Roles, data models, databases, network data model, relational data model

1. Introduction

1973 Turing Award winner Charles William Bachman is often credited as the first to have introduced the concept of roles into data modelling.[1] Bachman is a highly esteemed practitioner and a renowned expert in databases: he was the principal author of the first database management system, the *Integrated Data Store* (IDS, Bachman and Williams, 1964), and at the same time one of the main contributors to the CODASYL standard for the network data model. This data model, which allowed only non-recursive, 1:n-relationships to be represented directly, was strongly challenged by Codd's relational data model, which seemed more flexible since its relationships are generally m:n and unlike those of the network data model prescribe no paths for navigation.[2]

Bachman invested considerable personal effort in making the network data model fit for competition. One of his improvements was the introduction of alternate owner and multiple member records, which eventually led him to the *role data model*, a data model which Bachman liked to be understood as a generalization of the network and the relational data model (Bachman and Daya, 1977, p. 464). We all know how the rivalry ended: relational data base management systems have become industry standard, and another popular data model of that time, the entity relationship model (Chen, 1976), has persisted even into object-oriented modelling à la UML. However, the current popularity of object-orientation must give Bachman late satisfaction: in object-oriented programming, relationships are either 1:1 or 1:n, and the programmer is essentially a navigator. As a persistent extension of main memory, object-oriented

[1] Falkenberg's object-role model (Falkenberg, 1976) was in fact published earlier, but Bachman's practical work on roles goes back until at least 1973. Also, Falkenberg's use of the term role, although quite fundamental, was somewhat unorthodox: together with the term object, it served as a primitive based on which associations as well as object and association types were defined.

[2] "The programmer as navigator" was the title of Bachman's Turing award lecture (Bachman, 1973); he is commonly credited as having introduced the metaphor.

databases are also navigational, and to most object-oriented programmers interfaces to relational databases (embedded SQL, JDBC, and the like) are nothing but annoying anachronisms. The role concept, on the other hand, is of continuous interest even to this date, as best evidenced by this special issue.

With my modest contribution I try to reconstruct the development of Bachman's role data model and to view it from several different perspectives. I try to show that Bachman's original role concept is still viable today; that in fact it is perfectly natural, and extremely expressive at that. This is not to mean that it does not leave room for improvements, but rather that it can be viewed as an excellent starting point, one that deserves recognition beyond the usual "first mention" appreciation.

The remainder of this contribution is organized as follows. In Section 2, I will briefly resume the network data model and try to reconstruct how the role data model evolved out of it. A brief compilation of its most important properties follows in Section 3. In Section 4, I will show that Bachman's role concept is very general, and that it has been used, in more or less identical form, in disciplines much older than data modelling (or, should I rather say, much older forms of data modelling). In particular, I will argue that this generality, together with its minimality, deserves Bachman's role concept the status of an ontological primitive. Problems that remain can be fixed by strengthening the part of the relationship in data modelling, which is suggested in Section 5. I complement my contribution with refutations of two other popular definitions of roles (Section 6) and conclude in Section 7.

2. Evolution of the role data model

While today roles are a generally accepted notion in many computing disciplines, it seems that their need was first recognized by the database community. With hindsight, this should come as no surprise, since data modelling focuses on entities and the relationships between them; as we will see, roles are natural intermediaries between the two.

2.1. Background: the network data model

The network data model allows only binary 1:n-relationships, called *sets*, between entities (Elmasri and Navathe, 1989). One entity of each set (at the 1-end of the relationship) was called the *owner* record, the others (at the n-end) *member* records. Records represented entities by listing a number of attributes, called *items*. The main difference between the network and the hierarchical data model with its master-detail relationships was that a single record could participate in more than one set, the sets thus forming a network. The relationships of the network data model were statically typed (so-called *set types*) in that they had to declare the entity types occupying their owner and member places; such a declaration consisted of the name of the set type and the names of the owner and member types. An example of this is shown in Fig. 1(a).

Instances of a set type were commonly represented as ring structures (Fig. 1(b)). Because the ring structure carried no information as to the status (owner or member) of an element of a ring, each record carried a type field stating its record type (Elmasri and Navathe, 1989, p. 290). This way (and by knowing the set type), the status of a record in a set could be reconstructed. Hence, the network data model was statically typed, while at the same time each instance carried (runtime) type information, a feature commonly found with today's object systems.

Presence of instance type information was justified with hindsight by the later introduction of so-called *alternate owner* and *multiple member* (or *multimember*) *sets*, sets that allowed owners and members of the same set to be of different types (Bachman, 1969). Multiple member sets were necessary to

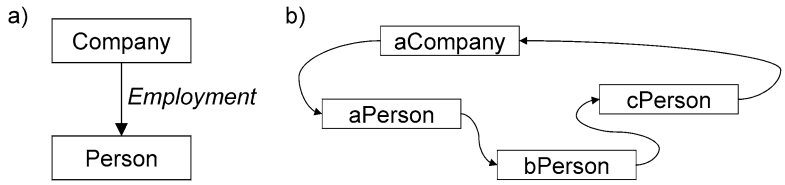

Fig. 1. The Network Data Model. (a) Sample data structure diagram using Bachman's own notation of the same name (Bachman, 1969). The rectangle labelled "Company" is the owner (record) type, the one labelled "Person" the member (record) type. Both would be called entity types in the entity-relationship data model (Chen, 1976). The arrow labelled "Employment" represents the set type, which corresponds to a 1:n-relationship type in the entity-relationship data model, directed from the owner to the member. (b) An instance (called set occurrence or set for short) of the Employment set type consisting of one instance (called record) of the owner type (labelled "aCompany") and three instances (records) of the member type (labelled "aPerson" etc.) The records are connected through pointers to form a ring structure. To reach a record, the programmer had to navigate through one or more sets.

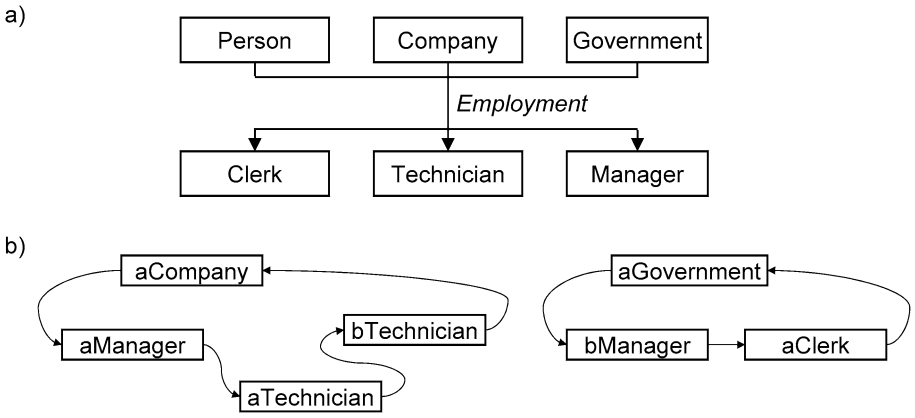

Fig. 2. Extension with alternate owner and multiple member sets. (a) Different owner and member types for the same set type. (b) Two set occurrences of the same set type, with records of varying types.

represent relationships such as Employment, where the member side could be occupied by instances of different record types such as Clerk, Technician, and Manager. Likewise, the owner side could be occupied by alternate types, for instance Person, Company, or Government (see Fig. 2). Today, these types are immediately recognized as subtypes of the (more general) Employee and Employer types, respectively, but the multiple member and alternate owner set concept was more flexible than subtyping (which is closely related to the notion of inheritance) in that it allowed instances of otherwise completely unrelated record types to replace for each other in the same set. And this is exactly where the runtime type information comes in: rather than treating all members as instances of an abstract type Employee or Employer (as today's object-oriented systems would do), in order to be able to process the member records of a set in a type-safe manner the processor had to be able to query their type and branch accordingly.[3] Bachman himself perceived this situation as annoying:

> The situation dealt with the processing of IDS chains (data structure sets) where there were two or more record types declared as members of the same set-type. When processing the members, one at a time, in response to FIND NEXT commands, it was necessary to immediately branch in the program, which processed the record retrieved, based upon record-name. This branching frequently

[3] Today immediately recognized as late binding!

> led to almost identical code which varied only because the items were accessed in each section by different record-name qualified items-names (DATE OF PURCHASE_ORDER vs. DATE OF SHOP_ORDER). This was always a nuisance because the program was bulkier and less readable than seemingly necessary. (Bachman, 1980, p. 10.)

At the same time, the possibility of having different types occupy one place of a relationship hampered formal comparison of the network data model with others, most prominently the relational data model, which offered no such possibility (Bachman and Daya, 1977; Bachman, 1980). This led to the dilemma that something that was needed to model reality more adequately and thus promised to make life of a coder simpler was difficult to map to what was known and in use at that time. An ideal setting for the introduction of a new construct.

2.2. Bachman's Eureka: role types

Bachman's first discovery was that

> ... most conventional file records and relational file n-tuples are in fact role oriented. These files typically deal with employees, customers, patients, or students, all of which are role types. (Bachman and Daya, 1977, p. 465.)

where a role, in Bachman's terms, is a

> behavior pattern which may be assumed by entities of different kinds. Furthermore, a particular entity may concurrently play one or more roles. Hence, the existence of all the roles of interest for a given entity characterize that entity. (Bachman and Daya, 1997, p. 465.)

His second important discovery was the following:

> When viewing the alternate owner declarations or multiple member declarations used with the network model, it appears that each has always represented one of two identifiable roles. The first role was played by the owners. The second role was played by the members. (Bachman and Daya, 1977, p. 466.)

Bachman's ingenious solution to the above described problem was then to give his data model

> the capability to declare a record type as having zero, one or more role-segment types. This facility permitted a role oriented item or item group to be accessed using role name qualification rather than record name qualification. (Bachman and Daya, 1977, p. 465.)

The role-segments required for the technical realization of his role concept were defined as follows:

> Within a data description, constructed according to the role model, the concept of a record description has been augmented by the concept of a role-segment description such that a record occurrence is a vehicle for one or more role-segment occurrences, each of a different role-segment type. Each record description consists of a record type name and a list of role-segment description references. A role-segment description may be referenced on one or more such lists. (Bachman and Daya, 1977, p. 466.)

Essentially, this meant that the relationship between entity types and role (segment) types was allowed to be $m{:}n$, as indeed required by his above quoted understanding of the role concept:

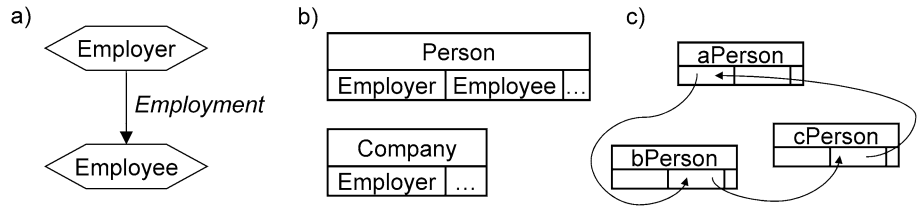

Fig. 3. Role data model. (a) The Employment set type relates to role types, called Employer and Employee. (b) Entity types such as Person and Company declare one or more role segments, here Employer and Employee. (c) Set occurrence with three Person entities in different roles.

> As the role concept became better understood it was obvious that there exists a many-to-many relationship between role types and entity types. [...] Some role types may characterize more than one entity type. For example, the employer role type may be associated with the entity types: person, corporation and government-unit. In a counter example, the role types: employee, customer, supplier, and stockholder may characterize the entity type person. (Bachman and Daya, 1977, p. 465.)

Finally, Bachman required that set types are defined on role types, not record types. This eliminated the need for alternate owner/multiple member types entirely:

> As in the network model, set relationships may be established with owner and member declarations. However, the alternate owner and multiple member record declarations will disappear. (Bachman and Daya, 1977, p. 466.)

As an additional constraint, the role data model required that owner and member roles be different (a condition that would be self-evident had role be interpreted as a distinguishing name for a place of a relationship, as in the relational and the entity relationship model).

The result of using the role data model for the example of Fig. 2 is shown in Fig. 3. Note that while initially an entity could still have items (attributes; not shown in the diagram) on its own, later all items where ascribed to roles. For the example this means that the person-related properties of an entity of type Person had to be represented by a special person role, a so-called *identity role* which Bachman admits to be "difficult to distinguish from the entity-type with which they are associated" (Bachman, 1980, p. 4).[4]

3. Properties of the role data model

Based on Bachman's description of the role data model, the following list of properties can be derived.

1. *The role data model defines relationships on role types, not entity types*. This is perhaps the most radical change Bachman suggested, and its consequences are far reaching. In particular, it almost invariably leads to an inflation of roles, since there needs to be a role type for every place of a relationship type (different relationship types may share same role types, though). Bachman already recognized this as a problem, and solved it by introducing the above-mentioned identity roles, role types that represent entity types exhibiting no other roles in relationships:

[4]This seems to be a technicality which is not generally useful for modelling (since it somewhat lifts the clear distinction between role and entity types). However, it makes the network data model a special case of the role data model (see below).

> However, it should be made clear that for many entity-types only one role-type, the identity role, is evident or generally of interest. In these cases, the Network data model, which does not discriminate between entity-types and role-types, is as useful as the Role data model. (Bachman, 1980, p. 4.)

2. *The role data model defines a m:n relationship between entity types and role types*. As pointed out above, a single entity type may support, i.e., provide entities for, several role types, and a single role type may be supported by several entity types. This is more general than other role models, in which roles can be played by only one type.
3. *The role data model distinguishes between existence of entities and their occurrence in roles*. Bachman separated the existence of an entity from its appearance in a relationship, where it exhibits role-specific behaviour. In his own words:

   > The Role Data Model [...] divided the object concept into two parts, a static part called an "entity" and a dynamic part called a "role". An entity established existence, while a role established behavior [5] for that entity. (Bachman, 1989, p. 30.)
   > An entity may now be accessed from several points of view, i.e., an entity with an 'employer' role is accessible independent of what else (person, corporation, etc.) it might be. (Bachman, 1980, p. 9.)

4. *The role data model treats role playing as statically declared, dynamically acquired*. Although the role playing capability of entities has to be declared statically (with their types), entities can adopt and drop roles dynamically:

   > If the entity represented by the record occurrence does not play all the roles declared for it, then the role-segments occurrences representing the missing roles would be present but would not be defined. The presence of a defined role-segment denotes the existence of the role for the entity. (Bachman and Daya, 1977, p. 466.)

   This has leads directly to the next property:
5. *The role data model introduces role types as supertypes of entity types*. The fact that role types replace sets of alternate owner and multiple member types in set declarations means that role types must be supertypes (or union types) of the types they replace: the role types must comprise all entities of all substituted entity types. [6] This is contrary to the view frequently held in the literature, according to which role types are subtypes (and thus, in case a subtype can have more than one direct supertype, intersection types) of the entity types providing the role players (Steimann, 2000). While this may seem plausible at first glance (a specific employee for example may be a person with additional properties and behaviour), it is actually false: if it were the case that role types are always subtypes, the role type Employer of Fig. 3(a) replacing the entity types Person, Company, and Government of Fig. 2(a) would actually have to be empty, since there is not a single entity that can be at the same time a person, a company, and a government. Instead, *all* persons, companies, or governments *can* be employers, and this fact is naturally specified in their type declarations. And

[5] Bachman seems to have adopted the term "behavior" from the above definition of roles as "behavior patterns", which was influenced by the use of the term in the theatrical context. I doubt that he meant behaviour in today's object-oriented sense, namely procedures or methods attached to objects (or their types). And yet, the different state associated with Bachman's roles (as captured by the items of the different role-segments) eventually leads to different behaviour of entities.

[6] As detailed in Steimann (2001), in object-oriented programming role types can be equated with interface types, which are also supertypes.

yet, at any certain point in time only some of the existing entities of the role-supporting entity types (here persons, companies, and governments) will actually play the role of an employer, i.e., belong to the extension of the corresponding role type. Bachman:

> However, not all of the entities of the entity type person will assume a role for each of the role types stated above. The purpose of the role model is to recognize and support formally this phenomenon which, once identified, is easy to recognize in the real world. (Bachman and Daya, 1977, p. 465.)

The seeming subtype/supertype paradox can be resolved by acknowledging the special character of the relationship between entity types and role types: while it expresses that all entities of all types supporting a role can play this role (or the entities of the types are not equal and therefore should be divided into distinct types according to their role-playing ability), at any point in time only that part of the entities that engage in a relationship in that role actually play the role. In other words: the extension of a role type does not only depend on the extensions of the entity types delivering role players, but also on the extensions of the relationships. As extensions of relationships breathe (grow and shrink) over time, roles are adopted and dropped accordingly (Property 4).

Note that later data models, for instance Elmsari et al.'s *entity category relationship model*, also allowed unions of types in the places of relationship declarations, but these were not called (and presumably also not considered to be) role types (Elmasri and Navathe, 1989, p. 421).

6. *The role data model offers polymorphism*. Defining roles as supertypes has an important corollary: by referring to role types rather than entity types, objects of different entity types can be treated equally, i.e., as if they were of the same type. In programming terms, this means that a variable declared with a role type can hold (and provide access to the features of) different entity types, without necessity of type casts or case analyses:

   > The motivation was to make it easier to program the navigation of alternate owner and multiple member set types. Conventional programming with record qualified item names requires continual inspection of the record occurrence retrieved to determine its record type. This determination of record type was necessary in order to select the branch in the program pertinent to the record retrieved. (Bachman and Daya, 1977, p. 465.)

   Today, this property is widely known as subtype or inclusion polymorphism, and is considered extremely valuable in programming. In fact, much of the power of object-oriented programming is associated with this property.[7]

   Note that because of the $m{:}n$-relationship of role types and entity types (Property 2 above), roles come with a dual, more literal form of polymorphism, namely that of a single entity having different forms. In fact, since entities can play different roles, and since each role may require different properties, a single entity can *appear in different forms* (the original meaning of the word polymorphism), namely one per role.[8]

7. *The role data model supports strong type checking*. This is another corollary of Property 5; it is best explained by Bachman himself:

[7] See also Kay (1993) for an interesting reflection on the origins of dynamic binding.

[8] By contrast, inclusion or subtype polymorphism refers to different objects having same form (with the effect that they can be assigned to the same variable declared with a single type): in this case, it is the properties associated with the type of the variable that are considered "polymorphic", since each object may implement them differently.

> The role model with its single owner role-segment type and single member role-segment type per set type makes the application of "type controlled" pointers a practical reality for set manipulation commands. Type controlled pointers are pointers which point only to objects of a specified type. This condition can be satisfied if the pointer type specification is role specific, because the first member and any next member of a set type are always the same role-segment type. Thus for languages which use type controlled pointers as an integrity feature, the role model offers a straightforward solution. The network model cannot because of the declarations of many record types as alternate owners or multiple members of a set type. (Bachman and Daya, 1977, p. 468.)

To summarize, a role in Bachman's role data model is a type that represents a partial view on entities as they participate in a relationship. All relationships are defined on role types, not entity types. An entity picks up a role by engaging in a relationship, and drops it by leaving the relationship. Entities of different types can play the same role and the same entity can play roles of different types. From the viewpoint of a relationship, all entities in one place of that relationship have the same role type and can therefore be treated alike in the context of the relationship, regardless of their possibly differing entity types.

4. Bachman's role concept as an ontological primitive

The word "role" is so omnipresent in our everyday conversations that most people find it hard to say what it means without using it. Etymologically, "role" derives from Latin "rotula" ("small wheel"), which is also the root of English "roll" and German "Rolle" (the latter translating to both role and roll in English). The metaphorical use of "rotula" and "Rolle", as well as the original meaning of "role", stems from theatre, where it denoted a roll (sic!) of papyrus on which the text for an actor was written. The term was then generalized to the part of the play itself, as which it denoted a protocol or behaviour specification an individual actor had to obey. Note that in classical plays, a role was never bound to a particular artist – in fact, it did not even require the right sex (in antiquity, female roles had to be played by males; actresses were unthinkable).

The theatrical meaning of role is also the one Bachman and Daya acknowledge as the source of their definition of role types:

> The use of the word role is taken from the theatrical context where a role is defined to be a part played by an actor on a stage. (Bachman and Daya, 1977, p. 465.)

But the database community was by far not the only one to adopt the term.

4.1. Roles in sociology

The theatrical use and meaning of the word "role" was soon transferred to the general interaction of humans in everyday life, and became occupied by sociology, the "science of society", quickly after its establishment as a discipline.[9] The following definition is taken from the Encyclopaedia Britannica:

[9]The first encyclopaedia I picked up to find out what sociology is about said that "role" was one of its two most important concepts. Unfortunately, at that time I still believed in the capabilities of my memory so I did not write down what the other was, nor did I note the encyclopaedia!

> role, in sociology, the behaviour expected of an individual who occupies a given social position or status. A role is a comprehensive pattern of behaviour that is socially recognized, providing a means of identifying and placing an individual in a society. It also serves as a strategy for coping with recurrent situations and dealing with the roles of others (e.g., parent-child roles). The term, borrowed from theatrical usage, emphasizes the distinction between the actor and the part. A role remains relatively stable even though different people occupy the position: any individual assigned the role of physician, like any actor in the role of Hamlet, is expected to behave in a particular way. An individual may have a unique style, but this is exhibited within the boundaries of the expected behaviour. [...]
>
> Role expectations include both actions and qualities: a teacher may be expected not only to deliver lectures, assign homework, and prepare examinations but also to be dedicated, concerned, honest, and responsible. Individuals usually occupy several positions, which may or may not be compatible with one another: one person may be husband, father, artist, and patient, with each role entailing certain obligations, duties, privileges, and rights vis-à-vis other persons. (Encyclopaedia Britannica.)

There are several interesting things to note about this definition. First, the relationship of roles and role players is generally $m{:}n$, i.e., the same person can play different social roles, and the same social role can be played by different persons. Second, a social role is defined in terms of its interaction with others, and is – to some extent – independent of the kind and properties of its role players. Although not explicitly stated, it should be clear that institutions and even computers can play certain social roles. In fact (and third), it seems that the relationship between a role and a role player is rather simple: from a static (i.e., atemporal) viewpoint, an entity must possess the capability (qualification) required by a role in order to be considered a (potential) role player, and from a dynamic viewpoint, at any point in time an entity either plays or does not play a specific role. (The quality of performance may of course vary.)

It appears that the concept of a social role has great similarity with Bachman's role concept. Therefore, mapping of roles identified in a social domain to role types of the role data model should come without "impedance mismatch", i.e., it should neither require nor introduce additional artefacts. This means that Bachman's role concept is not just a handy construction making the lives of coders and database administrators easier, but that in fact it is a natural extension of the conceptual repertoire needed to model (social) reality more adequately.

4.2. Roles in linguistics

The concepts used for data modelling always reflect some basic ontology, a list of concepts available to make a picture of the domain being modelled (also called "the furniture of the world" by Mario Bunge). Traditionally, these concepts include objects (or entities), attributes describing them, relationships linking them, and types thereof. These concepts seem to constitute a canonical set – after all, no less than predicate calculus, one of the best understood languages known to date, builds on them. However, predicate logic has no notion of roles.[10]

Predicate logic was devised as a formal variant of natural language, one that reduced language to its elementary constructs. Predicate logic and its predecessors have mostly been time ignorant (in the sense that they had no special means to express the time at which some assertion was true or false) – it should

[10]This ignores the meaning of roles as a named place of a relationship (Steimann, 2000), which can of course be added to predicate logic as syntactic sugar.

therefore come as no surprise that a concept as dynamic as that of a role had no place in it. However, that predicate logic is devoid of a role concept does not mean that it has no place in language.

In his quest for a universal language, the English merchant Francis Lodwick wrote a book describing his "Common Writing", "whereby two, although not understanding one the others Language, yet by the helpe thereof, may communicate their minds one to another". On pages 7–8 of this tiny book (of only 30 pages) Lodwick wrote:

> Next the verbes, follow in order the nounes substantives, of which there are two sorts.
>
> Appellative.
> proper.
>
> Appellative I thus distinguish. To be a name by which a thing is named and distinguished, but not continually, only for the present, in relation to some action done or suffered, as for instance, Speech being of a murther committed; he that committed the same, will, from the act, be called a murtherer, and the party on whom the act is committed, the murthered, these names thus given in reference to the action done, continues no longer with the party, then thought is had of the action done, but on the contrary the specificall proper name, remaineth continually with the denominated, as the specificall name of man, beast, so also the individuall denomination of any particular man, as *Peter, Thomas*, &c.
>
> A proper name is that, by which any thing is constantly denominated, specifically, as *Man, dog, horse*. (Lodwick, 1647.)

The first thing to note about Lodwick's remarkable work is that he placed the verbs (expressing predicates or relationships of a sentence) before the nouns, which is very much in line with modern theory of language (so called *dependency* or *valency theory*, according to which the objects of a sentence are governed by its predicates).[11]

The second thing to note is his distinction of two different kinds of "Appellatives": nouns like "murtherer" and "murthered" are temporary names of individuals, names that are defined in the context of and by a predicate or relationship, in this case the "murther committed"; whereas "specificall proper names" like "man" or "beast" present classifications that are independent of any situation, state, or relationship, and thus timeless. It is not difficult to see how the first category – specified by a relationship and serving as a temporary classification – corresponds to role types, while the latter corresponds to entity types.[12]

Lodwick's notion of a role was later rediscovered many times, for instance by the linguists Bühler and Fillmore (Steimann, 2002). Fillmore's semantic cases *Agent*, *Patient*, etc. are also called *thematic* (or *semantic*) *roles*; the recent tendency to move the grammar into the dictionary (Pustejovsky, 1995), where each word can specify for itself which others it may be combined with, builds on much finer grained roles as selectional restrictions.

It is interesting to note that the same pattern of definition of the role concept that was coined in theatre and that was transferred to the general social context seems to recur in linguistics: a role, so it seems, is a classifier for objects engaged in a certain place of a relationship, where classification lasts only as

[11] The focus on collaboration in object-oriented software modelling also seems to acknowledge this order.

[12] Note that modern English grammar distinguishes only between common and proper nouns, the former denoting objects anonymously ("man", "murderer"), the latter naming concrete individuals ("Peter", "Jack the Ripper"). Common nouns correspond to types, proper nouns to objects. However, no distinction between role types and entity types is made. It seems that this conceptual poverty has been readily adopted by logic and computer science, which traditionally also distinguishes between types and individuals (instances), but not between different kinds of types.

long as the objects takes that place. This is very much in line with Bachman's role type definition given above.

4.3. Independence from social domains

The prototypical roles recurred to in most of the literature are all roles of the same entity type, namely Person. This monotony begs the question whether roles and role playing are concepts whose applicability is restricted to social domains, or whether they can be used in other domains as well, including those where no persons are present.

The answer is simple: they can. For instance, a piece of paper can serve as input (playing the role Source) or output (with role Sink); Source can also be played by Keyboard, and Sink by Screen (but not vice versa). Although these examples are less intuitive, it should be clear that the role concept is legitimate wherever that of relationship is: the only point is that sometimes it may be perceived unneeded, or redundant.[13] However, this should not detract from the fact that all objects play roles whenever they participate in relationships – only sometimes, these roles may remain implicit.

4.4. Roles in ontology

If disciplines as different as data modelling, sociology, and linguistics agree on the need for, and to a large extent also on the definition of, roles as a concept, should this concept not also have a place in ontology? Furthermore, if *object*, *class*, and *relationship* are ontological primitives, does not *role* deserve the same status?

Among others, Nicola Guarino and co-workers have invested considerable effort in cleaning up with the confusion left by earlier, ad hoc ontological adaptations of role definitions. They base their definitions on two fundamental ontological properties, namely foundedness and (lack of) semantic rigidity, which serve to differentiate role types from other kinds of types (Guarino et al., 1994): roles are founded, since roles are defined in the context of relationships, and not semantically rigid, since entities can assume and drop roles without losing identity. Note that the same is true for social and linguistic roles as described above; it is certainly also true for Bachman's role concept. However, roles so defined are no more complex constructs than entities or relationships – they are indeed ontological primitives.

It is interesting to note that Bachman also regarded his role concept as primitive, i.e., as belonging to the fundamental repertoire of a language suitable to describe the world. Even if a reduction of this repertoire is possible, with it one loses semantic richness and thus naturalness of expression. To quote Bachman once more:

> The basic claim of the role model is that it more closely represents the real world than the network model or any other well known model. This better representation is made possible by the richness of the model. It exceeds these data models in its descriptive power. It is a model where the person describing the data can say more about the data and thus provides a better understanding of that data to the database management system. Thus a given amount of data may hold more information. (Bachman and Daya, 1977, p. 469.)

[13] This redundancy is best evidenced by the difficulty to find a good role name, one that is different from the entity type whose instances play the role. Cf. also Property 1 in Sections 3 and 5.

5. What Bachman did not tell us about roles

Despite the intriguing parsimony and elegance of Bachman's role concept, there is one thing he did not write of: that the same entity can play the same role more than once at the same time. For instance, a person can simultaneously hold several employments, with the same or with different employees. Each employment then comes with its own state, for instance an office telephone number, working times, and a salary. In fact, this observation is one of the strongest arguments in favour of the role-as-adjunct-object (or instance) representation (Steimann, 2000), according to which an object in a role is represented by a separate object adjoined to the entity playing the role. How can a definition of roles as supertypes, such as Bachman's, deal with this?

In the entity relationship model, and also in the relational data model, a relationship can itself carry data: it may possess attributes describing it. For instance, the Employment relationship can have attributes telephoneNumber, workingTimes, and salary. This lets different employments of a single person (in multiple employee roles) have different employment-related states, i.e., attribute values. All that is needed to ascribe this state to the person is some "relationship awareness" of the role player, i.e., knowledge of which relationships it participates in. This is granted by the network data model anyway; it may require a join in the relational data model (which is equally required for other queries of entity state).

Proponents of roles as adjunct objects may ask whether these attributes are natural properties of the relationship, or of the role. In case the latter seemed more natural, it could be argued that role-specific attributes should be detached from the relationship and ascribed to *separate role objects* distinct from both the relationship and the entities that fill its places. Theses objects would then act as bridges between the relationship and its related entities. If one feels that these role objects are closer to the entities than to the relationship, one indeed ends up with modelling roles as *adjunct objects*, by way of the *role object pattern* (see Section 6.1). However, as will be argued, this would deprive roles of their status as ontological primitives. Alternatively, one could view relationships as aggregations of role objects (and relationship types as combinations of role types, as for instance in Falkenberg's object-role model; Falkenberg 1976), but this would pose the question of how and where to define which roles have which counter-roles, i.e., which roles can legally be combined to form such an aggregation.

Another problem one might regard as not being adequately addressed by the role data model is that not all relationships define natural role types; that instead one may wish to be able to define a relationship between entity types directly (so-called *internals* such as whole-part or quality are such relationships; see Masolo et al., 2004). However, as mentioned above this problem was already foreseen by Bachman and is avoided in the role data model by the introduction of identity roles, roles that collect the properties of the entity type stripped of all (other) roles (cf. Section 2.2).

6. Alternative role data models

There are plenty of alternative definitions of roles and role modelling described in the literature. Some of these have been analyzed and discussed to some detail in (Steimann, 2000, 2002). Here, I will only briefly address two particular alternatives, because they seem to be so popular.

6.1. The role object pattern

The role object pattern (Bäumer et al., 1997) emulates the role concept through the primitives of object-oriented modelling, namely objects and links (in object-oriented programming usually realized

by pointers) between them (Fig. 4). Such an approach has many degrees of freedom, reflected in the many, slightly varying different implementations found in the literature, all of which have in common that they represent roles as *adjunct* objects or instances (Steimann, 2000). These approaches are known to cause a problem called *object schizophrenia*, but this can be considered a technicality that can be fixed by taking adequate measures in programming language design (for instance, by having two concepts of identity). Another problem is much more troubling.

The role object pattern and its siblings neglect the fact that the relationships involved in emulating the role concept themselves come with roles (e.g., the role player role and the role role in Fig. 4), and that this recursion lacks a self-contained resolution. To fully understand the problem, it is instructive to try and model the roles involved in the definition of the role object pattern using the role object pattern. The roles of the role data model on the other hand, as for instance the Owner and the Member role of a set (relationship), are defined without problems using the role data model as modelling language (Fig. 5).

Therefore, although the role object pattern (and also viewing roles as adjunct instances) has some appeal for practical reasons (because it can be fine-tuned to meet whosever intended semantics), it must be rejected for conceptual reasons, because it lacks the primitiveness one would expect from such a fundamental concept (cf. Section 4). In a way, emulating roles with objects and links is like representing both entities and relationships with tuples – the semantics of the construct, whether some expression denotes a role or something else, must be attached externally. Cf. also the comment of Bachman quoted at the end of Section 4.

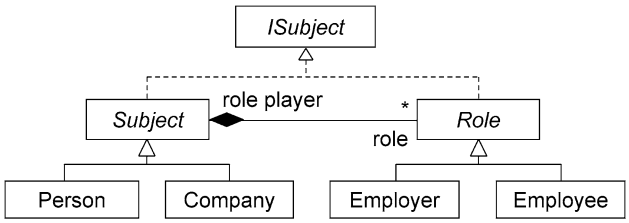

Fig. 4. Sketch of the popular role object pattern (Bäumer et al., 1997) as a UML class diagram. One abstract entity type (abstract class or interface in programming terms), called Subject, serves as the generalization (supertype) of the concrete entity types of the role players; another, called Role, as the generalization of the role objects. ISubject provides the common interface of the subject and its roles to clients (so that they can be used interchangeably in certain contexts). Note that the pattern has some superficial similarity with the alternate owner and multiple member types of Fig. 2(a), but the aggregation in the role object pattern (the connection of types Subject and Role) does not define a relationship between logically different entities (as is the case for the arrows in Fig. 2(a)), but one between a (core) entity and the same entity in different roles (thus the terms "roles as adjunct instances" and "object schizophrenia"). In particular, the connection of roles to other roles is missing. Note that the entity types Subject and Role themselves play roles, namely the role player and the role role (see text).

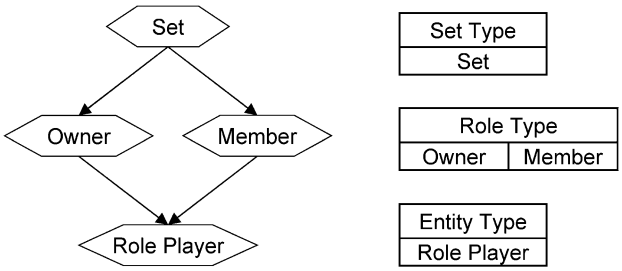

Fig. 5. The role data model defined in terms of itself. There are three entity types (Set Type, Role Type, and Entity Type), four role types (Set, Owner, Member, and Role Player), and four set types (unnamed). The notation is the same as in Fig. 3.

6.2. Roles as aspects

More recently, the role concept has been rediscovered in the context of what has become known as *aspect-oriented programming* (AOP, Kiczales et al., 1997). A role, so it is argued, is like an aspect in that it describes one particular facet shared by a number of objects of different types. However, roles and aspects differ in quite fundamental ways.

In brief, a role is a named type specifying a cohesive set of properties whose specification is determined by the collaboration with other roles and whose implementation by different classes is typically different (polymorphic). An aspect on the other hand is neither a type, nor is it meaningful only in the context of another aspect, nor does it introduce different implementations for different objects (it does in fact introduce same implementations, which is its very purpose). Although conceptually a role of an object can be viewed as an aspect of it, this aspect is typically not one in the aspect-oriented sense. A more detailed discussion of this can be found in (Steimann, 2005).

7. Conclusion

I am not sure why Bachman picked the term *role* for his new concept, but it seems like an obvious choice: like the terms *object*, *class*, and *relationship*, role is so fundamental a notion that it is hard to avoid it when talking about the world that surrounds us. So rather than wondering why Bachman chose the term *role* (and not *view*, *aspect*, or whichever others have), we should wonder why the concept had not been introduced to and used in data modelling before him. Be it as it may, there roles were, with a succinct definition and ready for use. That equivalent definitions had been in use in other disciplines such as linguistics and sociology only goes to show that it was wisely chosen.

Unfortunately, Bachman's role data model was refused the widespread recognition it deserved, so that its impact remained little. This was mostly accounted for by two other emerging data models: the relational data model, and the entity relationship model. It seems ironical that both of these come with their own notions of roles, but that these are much poorer in meaning than Bachman's: the relational data model defines roles as names used to distinguish places of a relation (or columns of a table) that happen to have the same type (and thus could not be distinguished by their type names); the entity relationship model, to which Bachman originally contributed,[14] similarly uses roles to distinguish the different lines connecting relationship types with entity types, facilitating readability where necessary and – again – differentiating repeated occurrences of the same entity type in a relationship type. Although choosing the term role for labels of the places of relationships is not unintuitive, it fails to acknowledge the semantic richness of the term as embodied in Bachman's role concept. This is evidenced by the huge number of alternative definitions of the role concept having been published to this date. Most of this work cites Bachman's assiduously, but only few authors seem to have grasped the fundamentality, naturalness, and simplicity of his original role definition.

In retrospect, Bachman himself describes the role data model as an episode, lasting no longer than from 1977 to 1980 (Bachman, 1989). After then, it seems that he had given it up in favour of what he called the *partnership data model* (Bachman, 1986, 1989), an attempt to rewrite the network and role

[14] Bachman, personal communication.

data models into something that still more closely represents the real world. However, this data model appears to come without an explicit role concept.[15]

To me, it remains unclear whether Bachman dismissed the role data model because he had lost the faith in the expressiveness of its major contribution, the role concept, or because he realized that its association with the network data model – confined to $1{:}n$-relationships as it was – would never allow him to regain the ground lost to the relational community (and pride would not permit him to transfer his role concept to the relational world). As you might suspect, I presume the latter was the case.

References

Bachman, C. W. & Williams, S. B. (1964). The integrated data store – A general purpose programming system for random access memories. *AFIPS Conference Proceedings (FJCC), 26*, 411–422.

Bachman, C. W. (1969). Data structure diagrams. *SIGMIS Database, 1*(2), 4–10.

Bachman, C. W. (1973). The programmer as navigator. *Commun. ACM, 16*(11), 653–658.

Bachman, C. W. & Daya, M. (1977). The role concept in data models. In *Proceedings of the Third International Conference on Very Large Data Bases (VLDB)* (pp. 464–476). IEEE Computer Society.

Bachman, C. W. (1980). The role data model approach to data structures. In: S. M. Deen and P. Hammersley (Eds), *Proceedings of the International Conference on Data Bases* (pp. 1–18). University of Aberdeen: Heyden & Son.

Bachman, C. W. (1986). Partnership data base management system and method. *US Patent No. 4631664* (http://www.freepatentsonline.com/us4631664.html).

Bachman, C. W. (1989). A personal chronicle: Creating better information systems, with some guiding principles. *IEEE Transactions on Knowledge and Data Engineering*, *1*(1), 17–32.

Bäumer, D., Riehle, D., Siberski, W., & Wulf, M. (1997). Role object pattern. In *Proceedings of PLoP'97*, Technical Report WUCS-97-34, Washington University, Dept. of Computer Science.

Chen, P. P. (1976). The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems, 1*(1), 9–36.

Elmasri, R. & Navathe, S. B. (1989). *Fundamentals of Database Systems*. Benjamin/Cummings.

Falkenberg, E. (1976). Concepts for modelling information. In G. M. Nijssen (ed.), *Proceedings of the IFIP Conference on Modelling in Data Base Management Systems* (pp. 95–109). Amsterdam: North-Holland.

Guarino, N., Carrara, M., & Giaretta, P. (1994). An ontology of meta-level categories. In J. Doyle, E. Sandewall, and P. Torasso. (Eds), *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)* (pp. 270–280). Morgan Kaufmann.

Kay, A. C. (1993). The early history of Smalltalk. *ACM SIGPLAN Notices, 28*(3), 69–95.

Kiczales, G. et al., (1997). Aspect-oriented programming. In *Proc. of ECOOP* (pp. 220–242).

Lodwick, F. (1647). *A Common Writing*. Reprinted in Salmon, V. (1972). *The Works of Francis Lodwick*. London: Longman.

Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., & Guarino, N. (2004). Social roles and their descriptions. In *KR 2004* (pp. 267–277).

Pustejovsky, J. (1995). *The Generative Lexicon*. Cambridge: MIT-Press.

Steimann, F. (2000). On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering*, *35*(1), 83–106.

Steimann, F. (2001). Role = Interface: a merger of concepts. *Journal of Object-Oriented Programming*, *14*(4), 23–32.

Steimann, F. (2002). *Ein natürlicher Rollenbegriff für die Softwaremodellierung*. Aachen: Shaker-Verlag.

Steimann, F. (2005). Domain models are aspect free. In *Proc. of MoDELS/UML 2005*, LNCS Vol. 3713 (pp. 171–185). Springer.

[15] Unfortunately, the partnership data model is not very well published – the only primary source that I found is a copy of a US patent (Bachman, 1986).