The Role Data Model Revisited

Friedrich Steimann

Fernuniversität in Hagen, Lehrgebiet Programmiersysteme
58084 Hagen, Germany
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 placement at the interface of entity types and relationship types, has always been underestimated. This is unfortunate since it has led to countless reinventions of the wheel and even regress. With this article, I hope to be able to shed some light on the naturalness of Bachman's role concept, and to make clear why I think that it is valid even today.

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), 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, which are generated on the fly, are generally *m*:*n* and 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 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 (Bachmann and Daya 1977, p. 464). We all know how the battle ended: relational data base management systems have become industry standard, and another data model of that time, the entity relationship model, has persisted even into object-oriented modelling à la UML. However, the current uprise of object-oriented databases 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 database are also navigational, and to an object-oriented programmer interfaces to relational databases (embedded SQL, JDBC and the like) are just annoying anachronisms. The role concept, on the other hand, is of continuous interest even to this date, as best evidenced by this symposium.

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 in excess of the usual "first mention" appreciation.

The remainder of this paper is organized as follows. First I will briefly resume the network data model and try to reconstruct how the role data model evolved out of it. A short summary of its properties follows. Next I will show that Bachman's role concept is rather universal, 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). I will argue that this universality, 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. I conclude my contribution with refutations of two other popular definitions of roles.

Evolution of the Role Data Model

Background: the Network Data Model

Deriving from the hierarchical data model, the network data model allows only binary 1:*n*-relationships. Originally, these relationships had to be declared as relationships between instances of two distinct types, the *owner type* (the 1-end of the relationship) and the *member type* (the *n*-end), where owner and member types were defined as Pascal-like records. A relationship type – called *set type* in the network data model – then declared the record types

[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 to define associations as well as object and association types.

[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.

in: G Boella, J Odell, L van der Torre, H Verhagen *Roles, an interdisciplinary perspective* AAAI Fall Symposium (2005).

of its owner and its members. A single record type instance (called *record occurrence*, or *record* for short) could be element (either owner or member) of several set type instances (called *set occurrences*, or *sets* for short), but not of the same set type. Hence, the network model could support only 1:*n*-relationships directly – *m*:*n*-relationships, as well as recursive relationships, had to be emulated. (Elmasri and Navathe 1989)

Instances of a set type were commonly represented as ring structures. Because the ring structure carried no information as to the status (either 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 strongly typed, while at the same time, each instance carried (runtime) type information, a feature commonly found with dynamically typed systems.

Instance type information was justified by the introduction of so-called *alternate owner* and *multiple member* (or *multimember*) *sets*, sets that allowed members to be of different types (Bachman 1969). Multimember sets (and set types) were necessary to represent relationships such as *Employment*, where the member side could be occupied by instances of record types such as *Clerk*, *Technician*, and *Manager*. Likewise, the owner side could be occupied by alternate types, for instance *Person*, *Company*, or *Government*. Today, these types are immediately recognized as specializations of the (more general) *Employee* and *Employer* types, resp., but the multimember and alternate owner set concept was more flexible than generalization (which inevitably comes with some 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 dynamic 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. Bachman himself perceived this situation (which he was largely responsible for) as a nuisance:

> 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 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 caused a dilemma, namely that something that was needed to model reality more adequately did not map to computing needs, neither theoretically nor practically. An ideal setting for the invention of a new concept.

Bachman's Eureka: Role Types

The first step to fix the problem was Bachman's idea to give the network 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. […] 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.[3] (Bachman and Daya 1977, p. 465)

Based on this technical consideration, which was implemented as early as in 1973, Bachman defined a role as 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)

The relationship between entity types and role (segment) types was soon recognized as being *m*:*n*:

> 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. 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)

This role concept was realized by so-called role-segments:

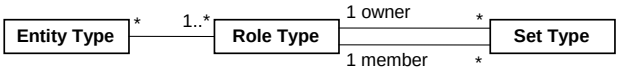
[3] The alert object-oriented programmer will immediately recognize the bad smell of the "replace conditional with polymorphism" refactoring (Fowler 1999).

> 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)

Sets (i.e., relationships) were defined on role types, not record types. Essentially, this eliminated the need for alternate owner/multiple member types:

> 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. 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. Therefore, the role model is defined with the constraint that […] only one role-segment description may be declared as the owner of a set description and only one role-segment description may be declared as the member of a set description. (Bachman and Daya 1977, p. 466)

The following UML class diagram may serve as a metamodel of the role data model:



While originally an entity could still have attributes ("items" in network database jargon) on its own, later all attributes where ascribed to roles (not shown in the metamodel). This meant that the person properties of an entity of type *Person* had to be represented by a *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]

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).

[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 for a more critical treatise.

Properties of the Role Data Model

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

1. *Distinction of Existence and Appearance.* Bachman separated between the existence of an entity and its appearance in a relationship. 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)

2. *Relationships Are Defined on Role, not Entity Types.* This has already been note above and is perhaps the most radical change Bachman suggested. Its consequences are far reaching, and it can lead to an inflation of roles (*cf.* last paragraph of "What Bachman did not tell us about roles" below).
3. *Role Types Are Unions of Entity Types.* Although not explicitly stated, the fact that different entity types could be collectively addressed using the same role type lets role types appear as unions (supertypes) of entity types. This is in sharp contrast to many views of roles held in the literature and – partly – also contrary to intuition, since roles appear to be *subtypes* of the role playing types. For instance, the *Employer* concept seems to be a specialization of the *Person* concept, since it adds properties to persons (resulting in a growing intension) and not all persons are employers (effecting to a shrinking extension). However, persons are not the only possible employers: companies and governments are other, and since these are non-overlapping (in the sense that their extensions are disjoint), *Employer* cannot be a common subtype of all three, because this would imply that its extension would always be empty. Also, although at a certain point of time only some persons may be employers, in principle all persons can be employer at some stage in their lifetimes, or all persons are not equal. It turns out that viewing roles as subtypes is based on a fallacy, namely on confusing the dynamic extensions of types with their static ones. See also Property 6.

   Note that later data models, as for instance Elmsari et al.'s *entity category relationship model*, also allowed unions of types in the places of relationship declarations,

[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) strict 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.

but these were not called (and presumably also not considered to be) role types (Elmasri and Navathe 1989, p. 421).

4. *Supportive of Strong Type Checking.* This is best explained by Bachman himself:

   > 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)

5. *Role-Based Polymorphism.* Defining roles as unions of types (Property 3) has an important corollary: by referring to role 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 of a single type can provide access to the features of different entity types, without type casts or explicit branching. This property is widely known in programming as inclusion polymorphism.

   As an aside, note that roles add another form of polymorphism, namely a literal one: since objects can play different roles, each one possibly requiring different properties, a single object can appear in different forms (one per role), the object hence being polymorphic in the literal sense (it takes on different forms). By contrast, with inclusion polymorphism the term refers to different objects having same form so that they can be assigned to the same variable. The objects are thus not polymorphic – it is their properties (mostly functions) that are.

6. *Role Playing is Dynamic.* Although the role playing capability of an entity type has to be declared statically, 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)

   Note that this duality, static role declaration and dynamic role adoption, leads to the somewhat paradoxical situation mentioned above: statically, all instances of all entity types declaring a role belong to the extension of that role, so that the role appears to be a supertype of the entity types. Dynamically, however, only some of all the entities of the role-playing entity types existing at a certain point in time belong to the extension of the role type. The resolution of this seeming paradox has to do with the fact that role types, their extensions in particular, cannot be defined independently of the relationship types whose places they are associated with. Attempts to provide relationship-dependent definitions of role types can be found in (Steimann 2000, 2002).

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 becoming member of 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 have the same role type and can therefore be treated alike, regardless of their possibly differing entity types.

Universality of the Role Data Model

Etymologically, the English word "role" derives from Latin "rotula" ("small wheel"), which is at the same time the root of English "roll" (both translating to the same German noun "Rolle"). The metaphorical use of "role" comes from theatre, where it denoted a roll 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 (including speech) an individual actor had to obey. In classical plays, a role was never bound to a particular person – in fact, it did not even require a fitting gender (in antiquity even female roles had to be played by male, since actresses were not allowed).

Occurrence in Sociology

The theatrical use and meaning of the word "role" was soon transferred to the general interaction of humans in everyday life, and – after its inception – quickly became occupied by sociology, the "science of society".[6] The following definition is taken from the Encyclopaedia Britannica:

> 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

[6] The first encyclopaedia I picked up said that "role" was one of sociology's 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!

> 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.

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.[7]

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.

Occurrence in Linguistics

The concepts of data modelling always reflect some basic ontology, a list of concepts that can be used to make a picture of the domain being modelled (the "reality"). 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.[8]

Predicate logic was devised as a formal variant of natural language, one that reduced language to its elementary constructs. However, that predicate logic is devoid of a role concept does not mean that it has no place in language. In fact, logic and its modern incarnation, AI, have a long tradition of ignoring time – it should therefore come as no surprise that a concept as dynamic as that of a role has been left aside. Linguists on the other had have not had such problems.

In his quest for a universal language, the English merchant 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) he 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, andc.
>
> 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). The recent focus on collaboration in object-oriented software modelling also seems to acknowledge this order.

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

[7] The quality of performance may vary.

[8] 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 a form of syntactic sugar.

serving as a temporary classification – corresponds to role types, while the latter corresponds to entity types.[9]

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.

Independence of Social Domains

The prototypical roles recurred to in most literature are all roles of the same entity type, namely *Person*. From this monotonicity of examples the question arises 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: it 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 relationship is: the only point is that sometimes it may be perceived unneeded, or redundant.[10] 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.

The Role Concept as an Ontological Primitive

That analogous role concepts are present in domains as different as sociology and linguistics makes roles a hot candidate for becoming accepted as an ontological primitive. If object, class, and relationship are, why should not role be?

Among others, Nicola Guarino and co-workers have invested considerable effort in cleaning up with the chaos left by ontological adaptations of ad hoc role definitions. They base their definitions on fundamental ontological properties such as foundedness and (lack of) semantic rigidity, which can be used to differentiate role types from other kinds of types (Guarino et al. 1994). As it turns out, Bachman's role concept is founded, since roles are defined in the context of relationships, and not semantically rigid, since entities can assume and drop roles without losing identity. As an aside, note that the same holds for the social and the linguistic role concept.

Important with all ontological definitions of roles is that they accept roles as being primitive, i.e., as belonging to the fundamental repertoire of a language suitable to describe the world. Although a reduction of this repertoire seems always 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)

Summary

To conclude this brief (and admittedly also rather superficial) investigation of universality, there seems to be a recurring pattern of definition of the role concept, one that is based on protocol specification in the context of relationships. A role, it seems, is a classification of an object engaged in a certain place of a relationship that lasts only as long as the object takes that place. In order to be able to fill that place, the object must be able to obey to the protocol (behaviour specification) associated with the role. Strikingly, this common understanding of roles is very much in line with Bachman's role type definition.

What Bachman Did Not Tell Us about Roles

Despite the convincing elegance of Bachman's role concept, there is one thing he did not think – or at least speak – 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-instance representation (Steimann 2000), and its opponents (including myself) tend to ignore this. However, this is just unrealistic. But how can the situation be remedied?

In the entity relationship model (and also to some degree in the relational data model) the relationship has the potential to bear additional information: it may have attributes further describing it. For instance, the *Employment* relationship could be attributed with *telephoneNumber*, *workingTimes*, and *salary*. This would allow different employments of a single person (in multiple employee roles) to come with different employment-related attributes. All that

---

[9] 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 adopted by data modelling, which also distinguishes between types and individuals (instances), but not between different kinds of types.

[10] 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.

is needed is some "relationship awareness" of the person, i.e., knowledge of the fact in which relationships it participates (which is granted in the network model anyway).

One might be tempted to ask whether these attributes are specific to the relationship or to the role. If the latter is the preferred answer, it could be argued that the role-specific attributes should be detached from the relationship and ascribed to *separate role instances* distinct from both the relationship and the entities that fill its places. Theses instances would then act as bridges between the relationship and its related entities. If one feels that these role instances are closer to the entities than to the relationship, one ends up with modelling roles as *adjunct instances*, by way of the *role object pattern* (see below). However, as far as I can see there is no practical need to do this, nor do good theoretical arguments exist.[11]

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, this problem is avoided in the role data model by the introduction of identity roles, roles (like *Person*) that collect the properties of the entity type stripped of all its roles (*cf.* Footnote 4). In fact, Bachman 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)

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.

The Role Object Pattern

The role object pattern (Bäumer et al. 1997) and its likes emulate the role concept through the primitives of object-oriented programming, namely objects and relationships (in object-oriented programming called links) between them. Such an approach has many degrees of freedom, reflected in the many, slightly varying different implementations found in the literature, all of which represent roles as *adjunct 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 language design (for instance, by having two concepts of identity). Another problem is much more worrying.

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 *Subject* role and the *Role* role), and that this recursion is devoid of a meaningful beginning. To 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 relationship (set), are defined without problems using the role data model as modelling language (*cf.* Steimann 2002, Fig. 4.15, for the case of UML).

Therefore, although I have great sympathy for the role object pattern for practical reasons (because it can be fine-tuned to meet whosever intended semantics), I must reject it for conceptual reasons, because it lacks the primitivity I would expect from such a fundamental concept. In a way, emulating roles with objects and links is much 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.

Roles as Aspects

More recently, the role concept has been rediscovered in the context of what has become known under the term *aspect-oriented programming* (AOP). A role, so the suggestion, is like an aspect in that it describes one particular facet of an object. 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 is 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 treatise can be found in (Steimann 2005).

[11] As an aside (and partly contradicting myself), it is interesting to note that allowing more than one role segment of the same type for the same entity record would have lifted the 1:*n*-relationship restriction of the role data model, allowing general *m*:*n* (including recursive) relationships. This is so because it would have introduced a (strongly typed) *m*:1-relationship between roles and entities, which combines with the 1:*n*-relationships induced by a set type declaration to a *m*:*n*-relationship. In fact, given the possibility of *m*:1-relationships between roles and entities the sets could even be reduced to pairs, since any entity can now appear (in different role instances of the same type) in as many sets of the same type as needed, both as an owner and as a member. Although today practically irrelevant (since no one would seriously attempt to rewrite a network database management system), it proves the conceptual power of the role concept.

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 describing the world.[12] 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 modelling before. Be it as it may, there roles were, with a clear definition and ready for use. That analogous definitions had long been in use in other disciplines only goes to show that it was wisely chosen.

Unfortunately, Bachman's role data model was refused the widespread recognition it would have 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 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 the same relationship type. Although choosing the term role for labels of the places of relationships is not unintuitive, it effects to a castration of Bachman's role concept.

That a role concept amounting to the name of a place in a relationship was not sufficient is impressively 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 initial 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 data models into something that still more closely represents the real world. However, this data model appears to come without an explicit role concept.[13]

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[14]. As you might suspect, I presume the latter was the case.

References

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., and Daya, M. 1977. The Role Concept in Data Models. In Proceedings of the Third International Conference on Very Large Data Bases, October 6-8, 1977, Tokyo, Japan. IEEE Computer Society 1977 VLDB 1977: 464–476.

Bachman, C. W. 1980. The role data model approach to data structures. In: Deen, S.M., and Hammersley P. Eds. Proceedings of the International Conference on DataBases, 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., and Wulf, M. 1997. Role Object Pattern. In Proceedings of PLoP '97. Technical Report WUCS-97-34. Washington University, Dept. of Computer Science.

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

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

Fowler, M. 1999. *Refactorings: Improving the Design of Existing Code.* Addison-Wesley.

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

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., and Guarino, N.: Social Roles and their Descriptions. In *KR 2004*, 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. 2002. *Ein natürlicher Rollenbegriff für die Softwaremodellierung*. Aachen: Shaker-Verlag.

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

[12] The disapproving reader may try to explain what a role is without recurring to it.

[13] 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).

[14] and pride would not permit him to transfer his role concept to the relational world