

Roles as Entity Types: A Conceptual Modelling Pattern

Jordi Cabot and Ruth Raventós

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona
email: {jcabot,raventos}@lsi.upc.es

Abstract: Roles are meant to capture dynamic and temporal aspects of real-world objects. The role concept has been used with many semantic meanings: *dynamic class*, *aspect*, *perspective*, *interface* or *mode*. This paper identifies common semantics of different role models found in the literature. Moreover, it presents a conceptual modelling pattern for the role concept that includes both the static and dynamic aspects of roles. A conceptual modelling pattern is aimed at representing a specific structure of knowledge that appears in different domains. In particular, we adapt the pattern to UML. The use of this pattern eases the definition of roles in conceptual schemas. In addition, we describe the design of schemas defined using our pattern in order to implement them in any object-oriented language. We also discuss the advantages of our approach over previous ones.

1. Introduction

Accurate and complete conceptual modelling is an essential premise for a correct development of an information system. Reusable conceptual schemas facilitate this difficult and time-consuming activity. The use of patterns is a key aspect to increase the reusability in all stages of software development.

A pattern identifies a problem and provides the specification of a generic solution to that problem. The definition of patterns in conceptual modelling may be regarded in two different ways: conceptual modelling patterns and analysis patterns.

In this paper, we distinguish between a conceptual modelling pattern that is aimed at representing a specific structure of knowledge encountered in different domains (for instance the *MemberOf* relationship), and an analysis pattern that specifies a generic and domain-dependent knowledge required to develop an application for specific users (for instance a pattern for electronic marketplaces). Authors do not always make this distinction. For example, to Fowler, in [8], patterns correspond to our conceptual modelling patterns while to Fernandez and Yuan, in [7], patterns correspond to our definition of analysis patterns. For a further discussion on analysis patterns see Teniente in [29].

The goal of this paper is to propose a conceptual modelling pattern for roles. A role is meant to capture dynamic and temporal aspects of real-world objects. There

are some dynamic situations from the real world that are not well suited just with the basic modelling language constructs. For example, when we want to model situations where an entity can present different properties depending on the context where it is used.

Although definitions of the role concept abound in the literature of conceptual modelling [2][4][5][8][13][24][25] a non-uniform and globally accepted definition is given. Roles are difficult to represent. They are not merely reified names for the participants in events. As we show in section 3, they can neither be represented as subtypes of other entity types even assuming multiple classification and inheritance. Rather, roles have their own characteristics that require them to be specified with a particular language construct in conceptual schemas.

We identify common semantics of role models found in the literature and present a pattern that fulfils them. The use of this pattern eases the definition of roles in conceptual schemas. Moreover, we also discuss the design and the implementation of conceptual schemas that use our pattern to facilitate their implementation in object-oriented languages. We adapt the pattern to UML [19]. As far as we know, ours is the first approach that allows the definition of roles by using the standard UML.

The rest of this paper is organized as follows: the next section presents the Role Pattern. Section 3 comments related work and compare it with our proposal. Finally, conclusions and further work are presented.

2. Roles as entity types Pattern

In order to describe the role pattern we adopt the template proposed by Geyer-Schulz and Hahsler in [11] to describe conceptual modelling patterns (called by the authors analysis patterns). They adopt a uniform and consistent format, in contrast to Fowler in [8] who uses a very free format for pattern writing. Geyer-Schulz and Hahsler stress that adhering to a structure for writing patterns is essential since patterns are easier to teach, learn, compare, write and use once the structure has been understood.

Their template preserves the typical context/problem/forces/solution structure of design patterns but adapted for the description of conceptual modelling patterns. The template includes the following sections: (1) Pattern Name. (2) Intent: what the pattern does and what problems it addresses. (3) Motivation: a scenario that illustrates the problem and how the pattern contributes to the solution in the specific scenario. (4) Forces and Context that should be resolved by the pattern. (5) Solution: description of all relevant structural and behavioural aspects of the pattern. (6) Consequences: how the pattern achieves its objectives and the existing trade-off. (7) Design and implementation: how the pattern can be realized in the design stage. (8) Known uses: examples of the pattern.

Note that, in the same way design patterns include the outline of possible implementations of the pattern [10], our conceptual modelling pattern includes the outline of the design of the pattern.

Following this template, next sections present the *Roles as entity types* Pattern.

2.1 Intent

The intent is the representation of roles that entities play through their life span and the control of their evolution.

2.2 Motivation

The role concept appears very frequently in conceptual modelling. However, the possibilities that offer conceptual modelling languages to deal with them are very limited and cover only a small part of the role features (see, for example, what UML supports in [6] and [27]).

There is not a uniform and globally accepted definition of roles. We illustrate here some of the most relevant ones:

- “It is a defined behaviour pattern which may be assumed by entities of different kinds”, Bachman and Daya in [1].
- “Role classes capture the temporal and evolutionary aspects of the real-world objects”, Dahchour et al. in [5].
- “Roles allow an object to receive and send different messages at different stages of evolution”, Pernici in [24].
- “Roles are founded; defined in terms of relationship to other things, and lacks of “semantics rigidity” (something is semantically rigid if its existence is tied to its class)”, Guarino in [13].

To summarize the above definitions, we could say that roles are useful to model the properties and behaviour of entities that evolve over time. The entity type *Person* is an illustrative example. During his or her life, a person may play different roles, for example he or she may become a student, an employee, a project manager, and so forth. Besides this, a person may have different properties and behaviour depending on the role or roles he/she is playing in a certain instant of time.

For instance, consider the following scenario: let Maria be a person who starts studying at a University (Maria plays the role of student). After some years of study she registers to a second university degree (Maria plays twice the role of student) and starts to work in a company (Maria plays the role of employee). In that company she may become a project manager (now Maria through her employee role plays the role of project manager). Note that, in this scenario, if we ask for the telephone number of Maria, the answer is not trivial since depending on the role or roles she is playing it may be her personal or her company phone number.

Taking into account the complexity of the notion of role and the lack of support for roles in present conceptual modelling languages, it is clear that a pattern to define such a common construct is needed in conceptual modelling.

2.3 Forces and Context

Our definition of the role concept is refined by describing the set of features that roles must meet, most of which have been identified by Steinmann [26]. In our case, these features are the forces that influence and should be resolved by the pattern.

We describe them using some examples over the scenario introduced above:

1. Ownership. A role comes with its own properties [15][5][14][30], i.e., an instance of *Employee* has its own properties which may be different than the ones of the entity type that plays such a role.
2. Dependency. An instance of a role is related to a unique instance of its entity type and its existence depends on the entity type to which it is associated to [15][4][5], i.e., it is not possible to have an instance of *Student* not related to an instance of *Person*.
3. Diversity. An entity may play different roles simultaneously [15][5][12][24][14][30][31], i.e., an instance of *Person* may play simultaneously the role of *Student* and *Employee*.
4. Multiplicity. An instance of an entity type may play several instances of the same role type at the same time [15][5][12][14][24][30][31]. For instance, a person that registers to more than a University have multiple instances of *Student* related to it.
5. Dynamicity. An entity may acquire and relinquish roles dynamically [1][15][5][12][14][22], i.e., a person may become a student, after some years become an employee, finish his/her studies, become a project manager, start another degree and so forth.
6. Control. The sequence in which roles may be acquired and relinquished can be subject to restrictions [5][24][30], i.e., a person may not become an employee when he/she is older than 65 years.
7. Roles can play roles [15][4][5][30][31]. This mirrors that an instance of *Person* can play the role of *Employee* and an instance of *Employee* can also play the role of *ProjectManager*.
8. Role identity [30]. Each instance of a role has its own role identifier, which is different from that of all other instances of the entity to which is associated with. This solves the so-called *counting problem* introduced by Wieringa et al in [30]. It refers to the fact that we need to distinguish the instances of the roles from the instances of the entity types that play them. For example, if we want to count the number of people that are students in a university (i.e. every person who is registered to at least a program in such university), the total number is less than the number of registered students in such university (in this case a person is counted twice if he or she is registered at two programs).
9. Adoption. Roles do not inherit from their entity types [15][12]. Instead, instances of roles have access to some properties of their corresponding entities i.e., *Student* may adopt *name* and *address* properties of *Person* but neither *religion* nor *marital status* properties. Therefore, the *Student* role cannot use the last two referred properties.

2.4 Solution

We divide the solution of our role pattern in two subsections. The first one deals with the structural aspects of roles while the second one deals with their evolution.

2.4.1 Structural Aspects of Roles

We believe there is not a fundamental difference between roles and entity types since roles have their own properties and identity. Therefore, we represent roles as entity types with their own attributes, relationships and generalisation/specialisation hierarchies. For practical reasons we call *role entity* types (or simply role if the context is clear) the entity types that represent roles and *natural entity types*¹ (or simply entity types) the entity types that may play those roles.

We define the relationship between a role entity type and its natural entity type by means of a *RoleOf* relationship. This special relationship relates a natural entity type with a role entity type to indicate that the natural entity type may play the role represented by the role entity type. In the relationship we also specify the properties (attributes and associations) of the natural entity type that are adopted by the role entity type.

Note that, since roles may play other roles, the same entity type may appear as a role entity type in a *RoleOf* relationship and as a natural entity type in a different *RoleOf* relationship.

Although this representation may be expressed in many conceptual modelling languages, in this work, we only adapt it to UML. In particular, we use UML 2.0 [19] and OCL 2.0 [18] versions.

To be able to represent the *RoleOf* relationship we use the extension mechanisms provided by UML, such as stereotypes, tags and constraints. Stereotypes allow us to define (virtual) new subclasses of metaclasses by adding some additional semantics. A stereotype may also define additional constraints on its base class and add some new properties through the use of tags.

The <<RoleOf>> stereotype allows us to define a *RoleOf* relationship between the natural and role entity types. The base class of the stereotype is the *Association* metaclass, which represents association relationships among classes. The <<RoleOf>> stereotype also includes the properties² the role adopts from the natural entity type. They are represented with a multivalued tag, called *adoptedProperties*. We may pack this stereotype in a new UML Profile [19] for Roles. Figure 1 shows the definition of the <<RoleOf>> stereotype.

¹ The *natural entity type* of a role relationship has sometimes been called *object class* [5][30]; **Error! No se encuentra el origen de la referencia.**, *ObjectWithRoles* [12], *natural type* [13] [26], *base class*[4], *entity type* [1], *entity class* [2], *base role* [23], or *core object* [3].

² A property in UML 2.0 [19] represents both the attributes and associations of an entity type.

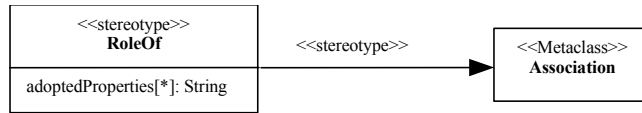


Figure 1. Definition of the RoleOf stereotype.

The multiplicity of the role towards its entity type is '1' (since a role can only be related to a single instance of the entity type) and its settability is *readOnly* (the role instance must always be related to the same instance of the entity type).

As an example, figure 2 shows the extended example introduced in section 2.2 specified in UML. The figure illustrates a natural entity type, *Person*, with its own properties, playing two roles: *Student* and *Employee*. The role *Student* is a generalisation of domestic and foreign students. The role *Employee* may play also the role of *ProjectManager*, who manages a set of tasks. *Student* adopts properties name, phone number and country (represented as attributes) and address (represented as an association) from *Person*, and *Employee* adopts the name and the derived age attribute. *ProjectManager* adopts name, employee number and the contract expiration date from *Employee*.

Note that *Employee* has its own phone number different from the *Person*'s phone number, i.e., *Employee* does not adopt the phone number attribute from *Person*. Therefore the answer to the question: "which is the phone number of Maria?" will vary depending on whether we are considering Maria as an instance of *Person* or *Employee*. The stereotyped operations shown in the figure will be taken up in the following section.

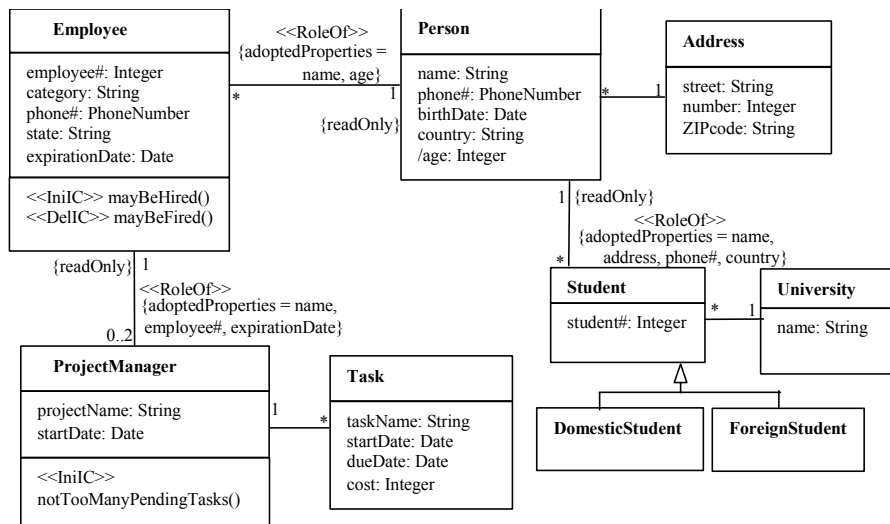


Figure 2. Example of *RoleOf* relationships in the UML

To complete the definition of the static aspects of roles we must attach some constraints to the `<<RoleOf>>` stereotype in order to control the correctness of its use. The constraints are the following:

- A stereotyped `<<RoleOf>>` association is a binary association with multiplicity '1' and settability `readOnly` in a member end
- Each value of the `adoptedProperty` tag must coincide with the name of a property of the natural entity type.
- A role entity type can only be related throughout a `RoleOf` relationship to at most a natural entity type.
- No cycles of roles are permitted; a role entity type may not be related throughout a direct or indirect `RoleOf` relationship to itself.

Adopted properties by the role from its natural entity type may be considered as implicit properties of the role entity type. Nevertheless, in order to facilitate the use of this adopted properties (for instance, when writing OCL expressions) we may need to include them explicitly in the role entity type. In this case, we add an extra property in the role entity type for each adopted property. These extra properties are labeled with the `<<adopted>>` stereotype to distinguish them from the own properties of the role entity type. In addition, they are derived. Their derivation rule always follows the general form:

context RoleEntityType::adoptedPropertyX: Type
derive: naturalEntityType.propertyX

Note that, to facilitate the work of designers, these added properties can be automatically generated. Figure 3 extends a subset of the previous example illustrating the `Student` role entity type including its adopted properties.

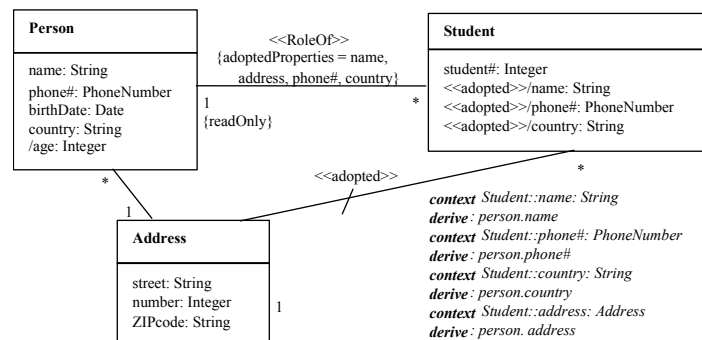


Figure 3. Example of the `Student` role entity type.

2.4.2 Role Acquisition and Relinquishment

So far, we have introduced a representation of the static part of the *Roles as entity types* Pattern. Nevertheless, this is not enough since role instances may be added or removed dynamically from an entity during its lifecycle and this addition or removal may be subjected to user-defined restrictions.

Since roles are represented as entity types we may define constraints on roles in the same way as we define constraints on entity types. Some of the constraints are inherent to our role representation (for example, that a person must play the role of *Employee* to play the role of *ProjectManager*, is already enforced by the schema).

Other restrictions involved may be expressed by means of the predefined constraints of UML. For example, to restrict that an *Employee* cannot play more than twice the *ProjectManager* role simultaneously, it is enough to define a cardinality constraint in the relationship. The definition of the rest of constraints requires the use of a general-purpose language, commonly OCL in the case of UML. For instance, we could specify OCL constraints to control that:

- A *Person* can only play the role of *Employee* if he/she is between 18 and 65 years old:
context Employee inv:
self.age >= 18 and self.age <= 65
- Any task of a *ProjectManager* must finish before his contract expires
context Task inv:
self.dueDate < self.projectManager.expirationDate

These OCL constraints are static, and thus, the role instances must satisfy them at any time. However, many of the restrictions that may be involved in the evolution of roles only apply at particular times, concretely they only need to be satisfied when the role is acquired or when it is relinquished. To specify such constraints we use the notion of creation-time constraints defined by Olivé in [17] and, in a similar way, we define the deletion-time constraints.

Creation-time constraints must hold when the instances of some entity type are created (in our case when the role is created). Deletion-time constraints must hold when the instances of some entity type are deleted (in our case when the role is deleted). These constraints are represented as operations, also called *constraint operations*, attached to the entity types and identified by a special stereotype. The creation-time constraint operations are marked with the stereotype <<IniIC>>. We define the stereotype <<DelIC>> for the deletion-time constraint operations.

These operations return a boolean that must be true to indicate that the constraint is satisfied. If the operation returns false (i.e., the constraint is not satisfied) then the creation or deletion event of the role is not accomplished. When appropriate, the operations are automatically executed by the information system.

As an example, we have defined the following restrictions in figure 2:

- A person cannot become an employee if he/she is studying two university degrees simultaneously. Note that this does not imply that a person that is already an employee may apply for two degrees.
context Employee :: maybeHired () : Boolean
body: self.person.student->size() < 2
- An employee may not be fired if he or she is in maternity leave.
context Employee :: maybeFired () : Boolean
body: self.state <> 'MaternityLeave'
- An employee may not become a new project manager if he/she still holds more than ten pending tasks.
context ProjectManager :: notTooManyPendingTasks(): Boolean
body self.employee.projectManager.tasks->
select(dueDate > Today)->size() <= 10

2.5 Consequences

Our pattern of roles achieves the objectives proposed in Section 2.3 since it fulfils the role features outlined before:

- Ownership. As roles are represented as entity types, they may have their own properties.
- Dependency. The cardinality ‘1’ with the tag {readOnly} ensures that all role instances depend on a unique instance of the natural entity type.
- Diversity. As the *RoleOf* relationship is an association, entity types may have many *RoleOf* relationships.
- Multiplicity. This is obtained by the cardinality at the *RoleOf* relationship.
- Dinamicity. Entities are related to their roles through an association. Thus, an entity may acquire or retract instances of a role many times.
- Control. The sequence in which roles may be acquired and relinquished can be subjected to restrictions.
- Roles can play roles. Roles are represented by ordinary classes. So, they can be participants of a *RoleOf* relationship.
- Role identity. As roles are represented as entity types, their instances have their own identifier.
- Adoption. The *adoptedProperty* tag of the *RoleOf* relationship allows the definition of this mechanism.

A trade-off that one may find in our representation is that we do not consider that roles cannot be associated to different natural entity types. We consider that this situation may be solved by defining a common supertype for all the natural entity types that play such role. For instance, if we need *Client* to be role of both *Company* and *Person* (understood as a physical person), we could define a common supertype for *Company* and *Person*, called *LegalPerson*, which plays the role of *Client*.

On the other hand, we do not allow roles to remain unconnected to any entity, as for instance, *Employee* understood as a vacant position not played by any *Person*. This approach is commonly used when considering roles just as interfaces. We discuss the limitations of this approach in Section 3.

2.6 Design and Implementation

There are some design patterns useful for designing and implementing roles in object oriented languages [8]. However, most of them are unable to deal with our proposed role semantics completely. A well-known pattern close to our role defined semantics is the *Role Object Pattern* [3]. This pattern is especially well suited for role implementation when roles are deemed as a specialization (or a kind of specialization) of its entity type (see Pelechano et al. in [23] as an example).

Nevertheless, this pattern is not entirely appropriate for designing our conceptual modelling pattern. We encounter two main problems in the *Role Object Pattern*. First, it uses a common superclass for all the roles of the entity type. In our approach, the roles are independent entity types so they do not need to present any common properties that justify this superclass. Secondly, all the roles are forced to have the

same inherited properties; it is not possible to define different adopted properties for each role.

This is the reason why we advocate here for an adapted version of this pattern that it takes into account our complete role semantics, including the adoption mechanism and the creation-time and deletion-time constraints.

Given a natural entity type and the set of its roles, we create a class for the natural entity type and a class for each role. We create a different relationship between the natural entity type and each of its roles. This relationship will be used to navigate from the natural entity type to its roles and vice versa. We add to the natural entity type two new operations *addRole* and *deleteRole* in charge of adding (deleting) roles to the natural entity after checking the creation-time (deletion-time) constraints. We could also add other useful operations when dealing with roles, such as *hasRole* or *getRole*.

The problem of the design of the adopted properties may be regarded as the same problem as designing derived information. In general, from a design and/or implementation point of view, there are two different approaches to deal with derived information. The attributes may be computed if they are calculated by means of an operation or may be materialized if they are explicitly stored in the class. In this case, for each adopted property we add an extra operation to the role class that returns the value of the property of the natural entity type. The operation accesses the property of the natural entity type navigating through the relationship.

Figure 4 summarizes our proposal. In figure 5 we apply the proposed design pattern to a part of the conceptual schema of figure 2. Note that *Employee* is both a role for the *Person* entity type and a natural entity type for the *ProjectManager* role, and thus, it presents both a reference to *Person* (as a role entity type) and the operations *addRole* and *deleteRole* (as a natural entity type).

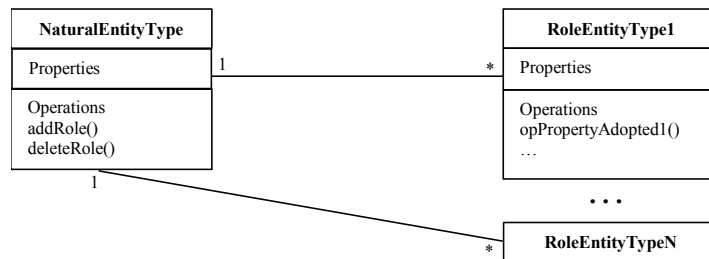


Figure 4. Summarized class diagram of the design.

This structure can be directly implemented in any common object-oriented language. As an example, in the Appendix, we show part of the implementation of *Person* and *Employee* classes in the Java language.

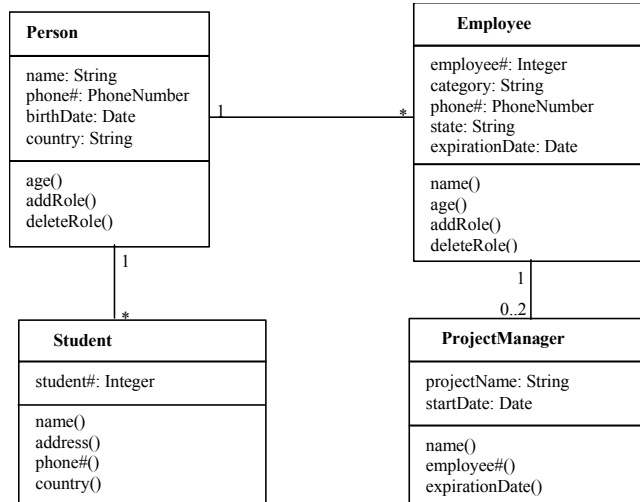


Figure 5. Example of an application of the design.

2.7 Known Uses

The role concept appears frequently in many different domains of the real world, since in each domain we can find entity types that present some properties that evolve over time.

Papazoglou et al. in [22] note that roles can be useful for several type of applications based on the use of object-oriented technology and they describe two examples of broad types of application that need role support: *security* and *workflows*. Some more examples are discussed by Jodlowski et al. in [14].

3. Related Work

Previous research can be grouped in four basic approaches to representing roles. We discuss the major drawbacks of each approach according to our role defined semantics. However they may suffice when considering more limited semantics.

The first approach represents a role as a label assigned to a participant in an event [19]. This representation does not achieve our objectives because roles come with their own properties different from those of the entity types playing them, which cannot be defined within the label.

A second approach considers that roles and entity types can be combined into a single hierarchy [1][4][25]. Role entity types are represented as subtypes of the natural entity type. For instance, if *Person* were a natural entity type, then *Student*, *Employee* and *ProjectManager* roles would appear as subtypes. Quite obviously, such a solution requires dynamic and multiple classification, since a person can change his/her role and play several roles simultaneously. However we would like to make

emphasis of three important features that specialization does not cover. First, what we have defined as multiplicity: an entity may play the same role more than once at the same time (i.e., specializations does not allow to define a *Person* playing simultaneously twice the role of *Employee*). The second one is adoption; with specialization we cannot restrict which attributes are adopted by the roles because they inherit all the attributes of their supertype. And finally, with specialization the role and the entity type have the same identifier, therefore the counting problem mentioned before is not solved. A further discussion on this topic can be seen in [26] and [14].

A third approach suggest that roles are only partial specifications of the entities playing them, and then the features of roles are the very features of interfaces (interfaces as types, in the sense of Java and UML) as Steimann in [28] or Lea and Marlowe in [16]. This alternative does not solve the multiplicity feature since an entity may not play the same interface more than once at the same time. On the other hand, roles do not have their own separated state (the whole state is shared in the natural entity type), since interfaces do not have their own attributes. Besides, when an instance of the natural entity type is created it acquires automatically all the roles, and thus, we cannot control nor restrict the evolution of the roles the instance of the natural entity type plays. Therefore, we consider that interfaces do not cover everything one might expect from the role concept.

The last approach, and also our approach, represents a role as a distinct element from an entity type but coupled to it [5][15][21][24][26]. However, most of these approaches use different semantics that the ones presented in this paper. For instance, some solutions are based on the fact that the instance of a natural entity type and its role instances share the same object identifier as Papazoglou et al in [21], among others. These solutions neither solve the counting problem mentioned before. Others, as Pernici in [24] do not allow roles to play roles. Our alternative suggesting roles as separated entity types fulfils the role semantics.

We believe one of the main advantages of our approach over previous ones is that we handle the complexity of role semantics in a very simple manner since we represent roles and its evolution with already existing elements (entity types and constraints) without adding completely new language constructs. Therefore, the designer can easily use the pattern to specify roles in conceptual schemas. In addition to this, our pattern describes a representation of roles in the standard UML, and thus, the pattern can be directly incorporated into current CASE tools.

We would also like to remark that our approach is complete and feasible in the sense that includes the design and the implementation of the pattern, in contrast to most of previous approaches that do not state how this could be achieved.

4. Conclusions and Further Work

This paper identifies the most important features of roles and presents the *Roles as entity types* pattern, a conceptual modelling pattern for roles. We have adapted the pattern to allow the specification of roles in UML conceptual schemas. To our knowledge, ours is the first standard extension to UML to define roles in conceptual

schemas in this language. The pattern can be easily implemented in any UML CASE tool in order to allow designers to use the role concept in their conceptual schemas.

The pattern includes the static aspects of roles as well as their evolution. We define roles as entity types (role entity types) related to natural entity types by means of a *RoleOf* relationship that includes the adoption of properties from the natural entity types by the role entity types. We have extended UML by means of the <<RoleOf>> stereotype to be able to represent such kind of relationships. To specify the role evolution we use two special kinds of constraints: creation-time constraints and deletion-time constraints. We have also discussed the design and implementation of conceptual schemas specified using the pattern

It would be interesting to study which taxonomies appearing in conceptual schemas should be better specified by using *RoleOf* relationships. This could be done by comparing the specification of the same case study with and without the use of roles. Moreover, we would like to automatize our approach by means of an application that given a conceptual schema (for instance, represented in XMI [20]) would generate automatically the corresponding classes in the target object oriented language. These are directions in which we plan to continue our work.

References

- [1] A. Albano, R. Bergamini, G. Ghelli, R. Orsini, "An Object Data Model with Roles", Proceedings of the 19th VLDB Conference. Morgan Kaufmann, 1993, pp. 39-51.
- [2] C.W. Bachman, M. Daya. "The Role Concept in Data Models", Proceedings of the Third International Conference on Very Large Databases, 1977, pp. 464-476.
- [3] D. Bäumer, D. Riehle, W. Wiberski, M. Wulf. "The Role Object Pattern", Proceedings of PLoP '97. Technical Report WUCS-97-34. Washington University Dept.
- [4] W.W. Chu, G. Zhang, "Associations and Roles in Object-oriented Modeling", Proceedings of the 16th Int. Conf. on Conceptual Modeling (ER'97), LNCS 1331, Springer, pp. 257-270.
- [5] M. Dahchour, A. Pirotte, E. Zimányi, "A Generic Role Model for Dynamic Objects", Proceedings of the 14th Int. Conf. On Advanced Information Systems Engineering (CAiSE'02), LNCS 2348, Springer, pp. 643-658.
- [6] R.Depke, G.Engels, J.M. Küster, "On the Integration of Roles in the UML", Technical Report No. 214, University of Paderborn, August 2000.
- [7] E. B. Fernandez; X. Yuan. "Semantic Analysis Patterns", Proceedings of the 19th Int. Conf. on Conceptual Modeling (ER'00), LNCS 1920, Springer 2000, pp. 183-195.
- [8] M. Fowler, "Dealing with Roles", PLoP '97 and EuroPloP '97 Conference, Technical Report #wucs-97-34, Dept. of Computer Science, Washington University, 1997.
- [9] M. Fowler, "Analysis Patterns: Reusable Object Models", Addison-Wesley, 1997.
- [10] E.Gamma, R.Helm, R.Johnson, J. Vlissides, "Design Patterns – Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994.
- [11] A. Geyer-Schulz, M. Hahsler, "Software Reuse with Analysis Patterns", Proceedings of AMCIS 2002, August 2002.
- [12] G. Gottlob, M. Schrefl, B. Röck, "Extending Object-oriented Systems with Roles", ACM Transactions on Information Systems 14 (3), 1996, pp. 268-296.
- [13] N. Guarino, "Concepts, Attributes and Arbitrary Relations", Data & Knowledge Engineering 8, 1992, pp. 249-261.

- [14] A. Jodłowski, P. Habela, J. Płodzien, C. Subieta, “Extending OO Metamodels towards Dynamic Object Roles”, R. Meersman et al. (Eds.): CoopIS/DOA/ODBASE 2003, LNCS 2888, pp. 1032–1047, 2003.
- [15] B.B. Kristensen, Object Oriented Modeling with Roles, Proceedings of the 2nd Int. Conf. on Object-Oriented Information Systems (OOIS’95), 1995
- [16] D. Lea, J. Marlowe, “Interface-Based Protocol Specification of Open Systems using PSL”, 9th European Conference ECOOP’95 - Object-Oriented Programming, LNCS 952 Springer 1995, pp. 374-398.
- [17] A. Olivé, “Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages”, Proceedings of the 22th Int. Conf. on Conceptual Modeling (ER’03), LNCS2813, 2003, pp.349-362.
- [18] Object Management Group, OMG Adopted Specification, “UML 2.0 OCL”, October 2003.
- [19] Object Management Group, OMG Adopted Specification. “UML 2.0 Superstructure Specification”, August 2002.
- [20] Object Management Group, “OMG XML Metadata Interchange Specification”, v.1.2, January 2002.
- [21] M.P. Papazoglou, B.J. Krämer, “A database model for object dynamics”, The VLDB Journal (6), January 1997, pp. 73-96.
- [22] M. P. Papazoglou, “Modeling Object Dynamics”, in. M.P. Papazoglou, S. Spaccapietra, Z.Tari (Eds.), Advances in Object-Oriented Data Modeling. MIT Press 2000, pp. 195-217.
- [23] V. Pelechano, M. Albert, E. Campos, O. Pastor, “Automating the Code Generation of Role Classes in OO Conceptual Schemas”, , Proceedings of the 4st Int. Conf. on Enterprise Information Systems (ICEIS 2002), 2002, pp. 656-686.
- [24] B. Pernici, “Objects with Roles”, Proceedings of the Conference on Office Information Systems, SIGOIS Bulletin, vol. 11, no. 2/3, ACM Press, New York, 1990, pp. 205-215.
- [25] J. Sowa, “Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley Publishing Company, New York, 1984.
- [26] F. Steimann, “On the Representation of Roles in Object-oriented and Conceptual Modelling”, Data & Knowledge Engineering 35, 2000, pp. 83-106.
- [27] F. Steimann, “A Radical Revision of UML’s Role Concept”, UML 2000: The Unified Modelling Language, LNCS 1939, Springer, pp. 194-209.
- [28] F. Steimann, “Role=Interface”, Journal of Object-Oriented Programming, October/November 2001, Vol. 14, Num. 14, pp. 23-32.
- [29] E. Teniente, “Analysis Pattern Definition in the UML”, Proceedings IRMA’2003, Idea Group Pub., pp. 774–777.
- [30] R.Wieringa, W. de Jonge. P. Spruit., “Using Dynamic Classes and Role Classes to Model Object Migration”, Theory and Practice of Object Systems, 1(1), 1995, pp. 61-83.
- [31] R. K. Wong, H. L. Chau, F. H. Lochovsky, “A Data Model and Semantics of Objects with Dynamic Roles”, 13th International Conference on Data Engineering, IEEE Computer Society, pp. 402-411.

Appendix A

```
public class Person
{
    public String name;
    public PhoneNumber phone;
    public Date birthDate;
    public Address address;
    Vector rols=new Vector()3;

    public double age() { //Age calculation}

    public void addRole(Object o) //Adding a new role
    {
        if (o instanceof Employee)
        { //Checking mayBeHired constraint
            int i=0; int numSt=0; Object o2;
            while (i<rols.size() && numSt<2)
            {
                o2=rols.get(i);
                if(o2 instanceof Student) numSt++;
                i++;
            }
            if(numSt<2) {rols.add(o);((Employee) o).naturalEntityType=this;}
            else System.out.println("Error 2St");
        }
        . . .
    }

    public void deleteRole(Object o)
    {
        if(o instanceof Employee) //Checking mayBeFired constraint
        {
            if(! ((Employee) o).state.equals("MaternityLeave"))
            { rols.removeElement(o); ((Employee) o).naturalEntityType=null;}
        }
        // ...
    }
}

public class Employee
{
    public int emp;
    public String category;
    public Object naturalEntityType;
    . . .
    //Adopted properties
    public String name() { return ((Person) naturalEntityType).name; }
    public double age() { return ((Person) naturalEntityType).age; }
}
```

³ Note that Person has a single multivalued attribute to store all the roles of that person, instead of having a different multivalued attribute for each of its roles (an attribute for the student instances, another for the employee instances...). We can use a single attribute since all the classes in Java are implicit subclasses of the class Object. When dealing with the attribute we make the appropriate castings to the specific role class.