

Extending OO Metamodels Towards Dynamic Object Roles¹

Andrzej Jodlowski¹, Piotr Habela², Jacek Plodzien^{1,3}, Kazimierz Subieta^{1,2}

¹Institute of Computer Science PAS, Warsaw, Poland

²Polish-Japanese Institute of Information Technology, Warsaw, Poland

³Warsaw School of Economics, Warsaw, Poland

andrzejj@ipipan.waw.pl

habela@pjwstk.edu.pl

jpl@ipipan.waw.pl

subieta@ipipan.waw.pl

Abstract. This paper discusses some of the implications of introducing the dynamic object role concept into object-oriented metamodels on both implementational and conceptual modeling levels in a coordinated way. The notion is expected to become one of the most fundamental constructs of an object data model and our research concerning object query languages allows us to state that it can be cleanly incorporated into existing object models. On the other hand the total cost of introducing the new notion needs to be considered. Thus, in this paper we look for a lightweight way of introducing object roles, depending as far as possible on already existing and popular concepts. Looking at the issue from the object database point of view, we discuss the necessary extensions and emphasize the importance of direct support of implemented notions from the means of conceptual modeling.

1 Introduction

Dynamic roles are a powerful mean of conceptual modeling that seems to fit especially well the object data model, by making it truly capable of expressing the interrelationships among real world entities. The notion is well known from numerous papers and a general agreement concerning its main conceptual features seems to exist. However, it still has not gained the level of acceptance that would allow standardization and a broader application.

Our research concerning the dynamic object roles [7, 15], allowed us to better understand a number of significant advantages of the new construct over the traditional means of modeling. We have also proposed the way of implementing the support for roles in object-oriented database management systems (ODBMS) and query language. However, despite the promising features, the popularization of the dynamic roles seems to be difficult due to their impact on an object data model. Both the power of this notion as well as the problems with it come from the fact it is neither

¹ Supported by the European Commission 5th Framework project ICONS (Intelligent Content Management System); no. IST-2001-32429.

just modeling feature nor just a programming construct. Instead, similarly like several other object-oriented concepts, it influences both modeling and implementation areas, which need to be considered together in order to make the notion truly productive. For those reasons, in this paper we look for the necessary data model and implementational environment extensions, taking into consideration the cost of their introduction and thus trying to align them as far as possible with already popular solutions. To establish a uniform view of the notion in both abovementioned areas, we attempt to model it as an explicit part of a common object-oriented metamodel.

One of the impediments is the additional complexity of a data model introduced by the new notion. Even today's object oriented metamodels on both modeling and implementation levels (see e.g. [12, 3]) are already very complex, and the cost of more powerful abstractions impacts the related tools, especially the DBMSs. All concepts of a chosen data model have to be implemented in DBMSs, effectively managed by their mechanisms and well understood by developers and administrators. Thus, any extension to such data model leads to a question if its benefits justify the additional complexity. For those reasons, a number of modeling notions that proved to be relatively complex while less essential (like e.g. n-ary associations) are not supported by implementation tools. This in turn makes them being rather seldom used by analysts and designers, who prefer constructs that can be more directly mapped into implemented structures.

The dynamic object role notion does not fall into this category. The first observation is that, if introduced, this would be a fundamental notion of a data model [13] rather than an extension or variation of some existing one. Secondly, as will be shown, it allows to cleanly solve a number of problems that occur with traditional approaches. Despite the notion is well motivated, the overall productivity of the proposed solution is strongly dependent on the way it is realized both in modeling and (especially) implementation. Thus, our assumptions are following:

- In order to benefit from already widely used solutions, it is desirable to introduce the notion with minimum extensions and impact on the existing concepts. For example, the amount of necessary changes and extensions of the UML metamodel can give an idea on the cost of particular proposal.
- The related implemented well-formedness rules (e.g. within a database schema) should provide some protection against programmers' errors, but on the other hand should not restrict the potential flexibility of the dynamic roles.
- The ability of the notion to support issues other than those of conceptual modeling, e.g. separation of concerns or metadata management, needs to be investigated.

The paper is organized as follows. In order to explain the starting point and justify the assumptions presented here, section 2 briefly presents the motivation behind introducing the dynamic object role as an explicit new notion (thus suggesting the necessary features) and provides a very brief summary of our approach to dynamic roles within an ODBMS, which has been discussed in [7]. Section 3 enumerates commonly used constructs that bear semblance of dynamic roles, providing the background of our discussion. Section 4 contains our proposals and considerations concerning extensions to the modeling language, ODBMS data definition language, database schema and query language. Section 5 concludes.

2 Dynamic Object Roles – Basic Assumptions

This section, based on our previous research, is introduced to explain the context of this work. The first subsection enumerates the main features of the notion, while the following section discusses the issues of modeling that may benefit from applying it. The third subsection summarizes our approach to the development of an ODBMS, including a pragmatically complete query language to manipulate its contents. Those two elements, that is, a modeling language and an ODBMS constitute a fairly complete framework, which allows us to analyze the total impact of the introduction of dynamic object roles.

2.1 Motivation Behind the Dynamic Role Notion

The situation where some entity can possess a number of different roles is ubiquitous in modeling. Although in some problem domains this may be not distinct (as they address only a certain aspect of a particular object's existence), in fact many occurrences of specialization relationship among classes constitute a (perhaps acceptable) simplification of the dependency that is in fact a dynamic role. For example, an information system of some university could model the class *Student* as the specialization of the class *Person*. Such a solution is practical only under the following conditions:

- Only one aspect of that person's existence (in this case – being a student) is within the domain of interest of the system. The other aspects (e.g. student's employment) do not need to be modeled.
- The system is not required to model a given person's data before or after the period of his/her being a student. That is, an object's migration from the *Student* class to some other class does not have to be supported.

If a certain object is described by several statically determined aspects (e.g. a vehicle classified according to its environment, powering system, purpose etc.), the traditional multiple inheritance can be used. However, with a larger number of such "dimensions" this approach results in a large number of rather artificial classes, some of which are not populated. Moreover, potential property name conflicts need to be faced. Thus, even in case of a truly static classification, the use of multiple inheritance can be problematic. If a system has to handle dynamically assignable roles (e.g. an *employee* role of person, a *project leader* role of an employee), then inheritance is not applicable: traditionally, association or aggregation is used to connect (a "base") object with objects describing its roles. This provides the necessary flexibility, however it is unnatural as a role is inherently dependent on its base object's identity and often also on its properties (including possible overriding). Thus the relationship of being a role lies somewhere between association and inheritance, while it is not satisfactorily describable by either of them.

In addition to the above problems relevant to conceptual modeling, the dynamic role notion can be also promising for some patterns and solutions at the design and implementation level. To sum up, the concept realizes the following features:

- Provides the expressive power of multiple inheritance, while avoiding name conflicts and combinatorial increase of the number of subclasses.

- Supports multiple-aspect classification and its maintenance by cleanly isolating code and data related with particular aspects.
- Allows for repeating classification (by having two or more roles of the same type).
- Covers the *variant* (or *union*) concept (known from e.g. C++ and CORBA): it can be achieved by modeling each variant branch as a role.
- Can realize object migration in a very flexible way: object's properties can be changed partly or completely (as if it were re-created under another class), while maintaining its identity.
- Provides a mean to realize the separation of concerns postulate in the spirit of AOP (Aspect-Oriented Programming). Each aspect can be encapsulated as a separate role, which would provide appropriate data and e.g. active rules.
- Similarly, it provides a convenient way to attach additional metadata to objects.

Thus, also a number of tasks that are more of technical than purely conceptual nature can be elegantly handled by roles.

2.2 Modeling With Dynamic Roles: Practical Applications

The need for the dynamic object roles is most distinct in complex systems, whose subjects have to be described in several different contexts. Another important reason is the need for dealing with complex lifecycles of objects. Although in that case using associations provides the necessary flexibility, this causes additional fragmentation and complicates data access.

An interesting and very distinct example motivating the use of dynamic object roles we have encountered in our work was a project for a Polish governmental institution dealing with regulations and investigations of the capital market. The investigations concerned various forms of data mining, making summary reports and checking legality of investors' operations on the market. Typical entities identified in the business domain were the following: person, broker, investment advisor, organizational unit, company, stock instrument, share, stock session, stock transaction, stock order, brokering house, document, legal regulation, and so on. The following characteristics of the system made it difficult to model using traditional notions (see also [15]):

- Some objects have many specializations at the same time. (e.g. a person being an employee and a broker simultaneously).
- Some objects have many specializations of the same type (e.g. a person being a member of many boards of directors at the same time).
- Some objects have specializations that depend on time. For instance, a person who was a broker a year ago, may currently be the director of a company. Furthermore, a person can be a broker several times, at different times and brokering houses.

The issue is not limited to people-related information. Similar problems, mostly related to recording historical information,² have also occurred with entities such as institutions, companies and documents.

² Despite this observation we do not assume the temporal information should be a predefined property of every object role, since the feature is rather domain-specific and can be explicitly modeled where necessary.

The above example may suggest the dynamic roles are especially applicable to the systems covering broad problem domains. However, taking into account that a database may serve several different applications, the need to model different roles of particular subject seems to occur quite frequently.

2.3 Assumed Implementation Model

In this subsection we attempt to summarize our approach to the development of an ODBMS, focusing on the features making it different from the popular solutions in the spirit of the ODMG standard [3]. The main concern of that research is the development of a full-fledged object-oriented query language equipped with imperative constructs, which makes it a sufficient mean of programming ODBMS-based applications. For this reason we do not consider integrating the roles with general-purpose programming languages. This solution allows to avoid the so-called “impedance mismatch” between a programming language and a query language. Another benefit is the ability to define a cleaner object model, independent on the design of those languages.

The following features of the proposed query language are important for its ability to incorporate the dynamic role notion:

- Any object (including sub-objects serving as attributes) is bound in a query by its instance name. The notion of *extent* is not used. A role, represented as an object can be treated analogously.
- The behavior of language’s operators and their scoping rules are precisely defined using the notion of environment stack. Particularly, the features of the static generalization hierarchy like inheritance and overriding can be easily described this way. The same mechanism can be useful for dynamic inheritance provided by roles.
- The presence of transitive closure operators makes it easier to process multi-level hierarchies, dynamic roles can form.

The language assumes a generic³ object store model, having the following properties (see Fig. 1):

- Any object has its own (not unique) name, a unique identifier and contents. Moreover (not shown in Fig. 1) it needs to indicate its type (if necessary, accompanied by class).
- An object contents can be either a collection of other (nested) objects, a primitive value (e.g. string, number, image) or a reference to an object.
- The instance name of every “root” object is added to the environment, so it can be bound analogously like a global variable and thus may serve as a starting point for querying.

³ It forms a base that does not prescribe any particular object model. Thus, appropriate metamodel needs to be defined over it. Important design decisions concerning e.g. the class mechanism and the features of dynamic roles have to be made there.

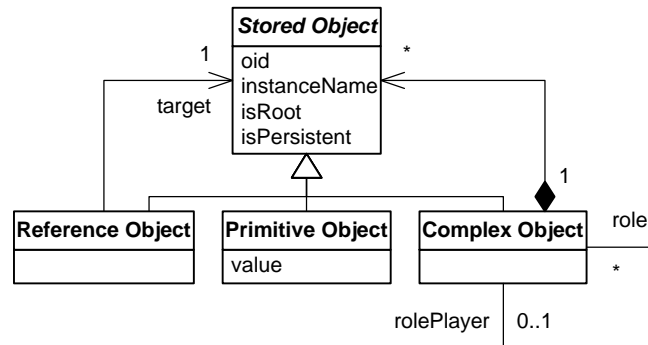


Fig. 1. A conceptual view of an object store model extended to support dynamic roles

As can be seen, to support dynamic roles, this model requires a new separate kind of relationship between objects, which can be called “is-a-role-of”. A more detailed discussion of the store model and language semantics extensions for dynamic roles can be found in [7]. While in such a model many important details remain open, we decided to hold the following assumptions:

- The “is-a-role-of” relationship can connect only complex objects. It can form pure hierarchies only (each role can have further own roles, but no cycles are allowed).
- Every role belongs to exactly one object. As a result it can *inherit dynamically* and *dynamically override* the base object’s properties, but it occurs only when the access is performed through this role. In other words, the query execution environment can differ, depending on the name of a base object or a name of one of its roles was used.
- The requirement for a role to be connected with its base object brings appropriate referential integrity constraints for copying and deletion operators.
- Since each role has its own unique identifier, it can be the target of reference from other object or an operation parameter.

This model served as a base for the implementation of our prototype system, consisting of an object store, schema repository and a query language interpreter.

3 Popular Solutions Related to the Dynamic Role Mechanism

Since we intend to fit the notion into traditional object data models, this section provides an overview of popular solutions related to the dynamic object role concept. On the other hand, the summary of the research concerning the dynamic object roles has been intentionally skipped here.

The case of a dynamic object role in modeling can be described as a single object having different sets of properties connected with particular aspects of its existence. As mentioned, in case of a multiple yet static classification, the most straightforward of traditional approaches to address it would be a multiple inheritance. At the same time however, this solution bears the largest number of problems and limitations:

- Possibility of name conflicts among inherited features.

- Inability to dynamically reclassify object.
- Inability to describe more than one role of the same type (e.g. instance of a class *Employee* or *Student-Employee* can not store information on a person's employment in two different companies).
- Combinatorial growth of the number of classes.

Note that in absence of multiple inheritance the situation is even worse, since code reuse is limited. That is, it becomes necessary to create unrelated “copy & paste” class definitions, instead of the natural code reuse through class specializations.

The above remarks confirm that mixing different aspects of a given object's description into a single class is not a proper approach. The conceptual modeling notions offer in this case more flexibility. The UML (Unified Modeling Language) [12] allows for specifying different specialization hierarchies for each criteria and even marking some of them dynamic. Such multidimensional classification results in a much clearer model. However, this part of the UML specification is not very detailed and, what is the most important, implementation tools do not support such features, thus the model loses its clarity during implementation.

Another notion bearing some of the desired features are Java's inner classes. Such a class is defined within the scope of its outer (base) class and its instances possess links to their base objects (being the instances of the outer class). Thus such an inner class can access properties of its base class in a way similar as if it were a subclass. Instances of an inner class can be referenced either from inside of a base class's object (that is, being its attributes) or anywhere outside its base object (in the latter case making it unaware of its dependent instances). Thus the following properties, resembling the dynamic object role concept, can be noted:

- Inner class can encapsulate some distinguished parts of object properties.
- Inner class's methods have access to the state and behavior of the base object as if they were defined in the outer class or its subclass.
- Arbitrary number of inner class instances connected with a given object can be created or removed during runtime.
- Instances of inner class connected with a given object of outer class are separated from each other, so no name conflict can occur.

As can be seen, the construct has many interesting features and can be useful in partitioning and encapsulating object's properties. However, this notion is quite far from widely agreed features of dynamic object roles in the following terms:

- There are no means of reassigning an inner class object to be connected with another outer class's instance.
- Creating an inner class definition requires access to the code of the outer (base) class. This breaks the “Open-Closed” principle [10] concerning class design.
- Private properties of an outer class are accessible to inner class's methods, which is probably undesirable in case of an object role.
- The inner class's access to the properties of outer class does not provide the former with interface of the outer class. Thus there is no substitutability between outer and inner class's instances.

Despite those limitations, the notion of Java's inner class seems to be the nearest to the dynamic object role concept from among of the broadly known solutions.

Other interesting features come from the AOP [9] language extensions. Its mission is to deal with so-called *crosscutting concerns*, being the features of software, which are difficult or impossible to modularize using classes, methods or procedures. They are often related with various kinds of non-functional requirements, like e.g. persistence, security, synchronization, real-time constraints or logging, whose implementation is usually scattered among different fragments of code, which makes it less readable and complicates its maintenance. The solution offered by the AOP is an abstraction called *aspect*, used to modularize a given concern in a way that does not affect the base code. This provides a kind of additional design “dimension”, which becomes integrated into the application during the compilation phase, by a mechanism called *aspect weaver*. Particularly, the well known Java-based implementation of the AOP paradigm, called AspectJ [1], provides two general means of isolating such properties:

- **“Pointcut” and “advice” declarations** modify certain elements of program’s control flow by augmenting or bypassing the statements of specified kind.
- **Class structure modification using *introduction***. This mechanism allows for modifying definition of a given class, both in terms of behavior (new or overridden methods) as well as the structure (additional attributes). This includes possibility to modify inheritance hierarchy by making existing class extend other class or to implement a certain interface. All these changes can be achieved by declarations located outside the considered class.

The latter feature allows to isolate properties introduced to a class design on behalf of certain aspect or role of object’s existence. However, it is necessary to note that this mechanism is purely static and, although very powerful and advantageous in terms of maintenance, it is not suitable for implementing dynamic object roles. Its capabilities in this area seem to be analogous to static multiple inheritance.

Although the modeling languages tend to be richer in terms of provided notions than implementational tools, the UML does not support a notion similar to the dynamic object role. It uses the term “role” in several other meaning; most commonly to describe an object’s participation in association link or its position within a *collaboration* of objects. The necessary semantics of dynamic object role can be neither expressed precisely nor conveniently without extending the language.

As can be seen, despite there are a number of already adapted similar solutions, the notion of dynamic object role is not satisfactorily addressed, which is the case both in modeling and implementation. It is currently implemented and even modeled in an indirect and rather low-level way (see e.g. [5]).

Considering the fundamental importance of that notion, it is possible to conclude that the dynamic roles should be introduced as a separate new construct both into modeling and into implementation languages and the DBMS area.

4 Extending the Framework to Incorporate Roles

In this section we discuss the features of a modeling language, metadata management and a query language, necessary to support roles. The following issues are considered:

- Augmenting the conceptual view of ODBMS metamodel with the role-specific elements and necessary constraints. This directly concerns the shape of a data definition language and indirectly – also the features of a query language.
- Providing conceptually compatible support from the side of a modeling notation.
- Identifying new patterns that can be realized using the introduced notion.

4.1 Augmenting the DBMS Metamodel with the Role Notion

It is necessary to note that the issue of an ODBMS metamodel brings a number of specific problems that do not occur e.g. in modeling languages' metamodels and thus the metadata structure like this suggested in the ODMG standard [3] may need to be significantly redesigned to successfully meet all the requirements related to a database schema (see [6] for detailed discussion). However, the problem does not impact the conceptual view of the core elements of the object data model, so we can assume the ODMG and UML metamodels with their popular understanding of the notions of class, association, attribute, operation etc. to be an appropriate starting point for our discussion.

As mentioned, it is usually assumed that roles should be able to complement all kinds of properties the base object possesses (e.g. attributes, association links, behavior (operations), as well as the ability to have their own roles). This requires supporting them with a specification similar to that of a regular object. That is, an interface specifying role properties and a class implementing them are necessary. Additionally, the role type definition brings an additional constraint: role needs to be connected with its base (also called “player”), which can be a regular object or another role. Moreover, if a role is treated as a property a base object is aware of, a role multiplicity (like in case of attribute or association link), as well as applicability to exactly one base class (the latter constraint is called “player qualification” in [16]) can be considered. Following this path requires very little change to existing metamodels. Taking the UML metamodel as an example of the mainstream object data model, we would suggest the following extension as shown in Fig. 2:⁴ *RoleClass* becomes a special kind of *Class*, distinguished by the fact that its instances cannot be instantiated in separation from their base objects.

An important question concerning constraints imposed on the dynamic object role construct is if a schema really should specify (and allow) exactly one class as a type of a base object for each role specification. This constraint seems to be justified by the following reasons:

- A role is indeed usually defined to extend the properties of exactly one object type. If there are more related types, they should have something in common, so their generalization could be used as a role's base.
- Operations provided by a role interface can have their implementation dependent on particular properties (operations, attributes and association links) of the base object.

⁴ Analogous properties were introduced in the data definition language (DDL) of our prototype implementation. The only difference was, that following the ODMG terminology the base for our extension was the notion of *Interface*, not *Class*.

- Since a reference to a role is intended to support its base object's features, assuring that such inheritance will take place requires appropriate type constraint.

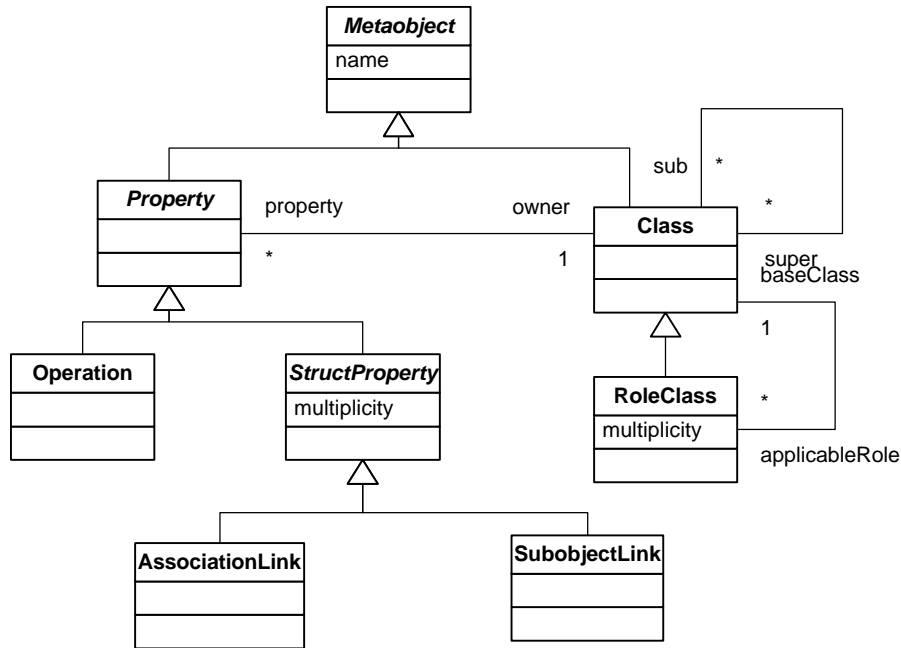


Fig. 2. Role concept incorporated into the conceptual metamodel (fragment). Note in this view a structural similarity of traditional static inheritance (sub-super relationship between classes) and dynamic inheritance (realized by a role).

On the other hand, also more “loosely” connected roles can be considered. Such roles would make no assumptions on their base objects’ types and serve as “handles” or “labels” for a number of otherwise unrelated objects. This brings another question: assuming we would allow defining roles that make no assumptions on their base objects’ properties, is the base class specification necessary for them? Summing up, this would lead towards much less restrictive typing constraints concerning the roles or to the need to distinguish two flavors of object roles.

Fortunately, we can achieve a satisfactory flexibility of the introduced notion while keeping the seemingly restrictive constraint of only one base class allowed, thanks to the following solutions:

- A single-rooted class hierarchy can be assumed (as e.g. in case of Java language, where all classes inherit from a predefined class *Object*). In that case all universally applicable roles can be specified as the roles of such root class.
- If some role can be applied to several specific base classes having no direct generalization, the properties of such role can be grouped within an abstract class, and specific role classes for each base can be defined as specialization of this abstract class (see Fig. 3; notation is described in the next subsection).

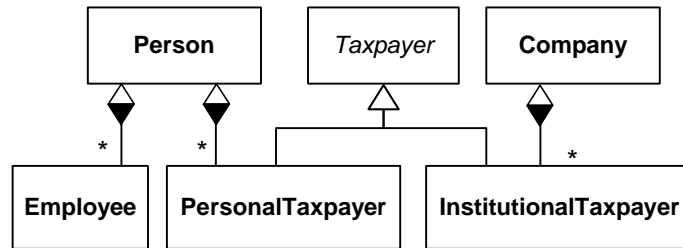


Fig. 3. Dynamic role dependency mixed with static inheritance (the details of notation are described in the next sub-section)

As the example (Fig. 3) suggests, the dynamic inheritance realized by roles can be freely combined with static inheritance. Two intuitive constraints seem to be necessary though:

- The combination of static and dynamic inheritance should not form cycles. Although there could be some interesting and perhaps useful effects of such structure, it does not seem worth of additional complexity and possible confusions.
- A regular class cannot be a specialization of role class. That is, the restriction of not having independent instances (assumed for a role class) is a constraint that should be inherited.⁵

4.2 Supporting the Role Notion in a Modeling Language

This task could be relatively easy, assuming the extensions for ODBMS and modeling language metamodel are coordinated. A natural choice for supporting notation here is the UML and the extension of its metamodel can be performed exactly as suggested in the above subsection. In that case we can benefit greatly from the postulate of possibly lightweight extension: since the *RoleClass* specializes UML's *Class* concept, we can reuse the class notation for role-classes and object notation for roles (Fig. 4). No special symbol for a *RoleClass* is necessary, since the presence of role dependency relationship distinguishes it from a regular class.⁶ We only need to introduce a unique symbol to indicate the "is-a-role-of" relationship at the object level and "role dependency" (shows a given role's applicability to objects of particular class) at the class level. Similarly like in case of UML's *aggregation* or *composition* we can use the same symbol on both levels.

The symbol of the "is-a-role-of" relationship was intentionally chosen to resemble both generalization (triangular arrow) and composition (filled diamond) [8] notation of UML, since the role relationship is characterized by the mix of their properties:

⁵ For example, if *Employee* is a role of *Person* and *Employee* is specialized into *Clerk*, then the *Clerk* class can only be instantiated as a role of some person, not as an independent object.

⁶ The only problem with *role class* identification occurs if a given diagram does not show all the classes and skips some role's base class. In such case we suggest marking the role class with the "is-a-role-of" relationship symbol with ellipsis on its "base" end.

- The way of overriding or augmenting the base object’s properties by its role bears semblance to static inheritance and is similarly realized within the stack semantics [7].
- The requirement for role to be connected to exactly one player as well as the transitivity of the “is-a-role-of” relationship, and e.g. the deletion propagation from player to its roles, make the relationship similar to UML’s *composition* with multiplicity fixed to “exactly one”. Thus, we need only one multiplicity specification for each role dependency, namely the one referring to the multiplicity attribute of *RoleClass*.

Another benefit of maintaining a straightforward mapping between the UML metamodel and the conceptual view of an ODBMS metamodel would be easier integration of the latter with the UML-inspired Meta Object Facility (MOF) model, being the central element of the Model Driven Architecture (MDA) initiative [11].

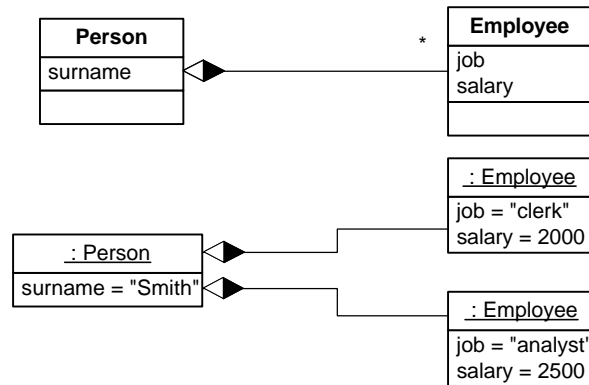


Fig. 4. Examples of applying the proposed extended UML notation to model dynamic object roles on the class level (role dependency relationship) and the object level (the “is-a-role-of” relationship)

4.3 Dynamic Roles: The Impact of a Data Model Extension

When considering the design of the role mechanism and weighting the related costs, it is worth to remind a very important idea of programming languages, known as “the principle of correspondence” [2]. It can be described as follows: any new feature introduced to the data model, requires considering its influence on binding mechanisms and its composition with other constructs of the language. This remark makes it clear, that extending today’s already complex object models can be very costly. On the other hand, it can be significantly reduced thanks to the “reuse” of already present features. For those reasons we choose to make possibly small extensions to existing traditional metamodel.

The benefits of such approach could already be observed in case of the modeling language. As shown in the above subsection, we were able to achieve the necessary expressive power by introducing only one new symbol, taking advantage of some similarities with already present relationships (namely – generalization and

composition). Although a number of additional properties can be considered (e.g. mutual exclusion of some roles connected to the same object), they do not seem to be worth the additional complexity. Analogous situation occurs for the DDL, since in our approach its statements need to have similar information content.

The similarities of the “is-a-role-of” relationship with the notions of generalization, association or UML’s composition are very useful, however following them too closely may be misleading. We have identified the following differences:

- In contrast to associations the “is-a-role-of” relationships may not form cycles.
- Also the “role dependency” relationships may form cycles neither alone nor in combination with the static generalization. Such a structure is problematic and does not seem to be useful.
- Despite the inheritance of base object’s properties, substitutability between a role and its base object does not occur. An attribute of a base object may be overridden by an attribute or link of the same name but of different type, defined in a role class.
- Method polymorphism is not extended over the dynamic inheritance provided by object roles. In other words, if an object is bound by its own name, its behavior is not influenced by the properties of its roles.

Although the abovementioned rules apply only to rather very specific situations, the difference between static “generalization-specialization” and roles needs to be well understood. As the example below (Fig. 5) illustrates, the “is-a-role-of” relationship indeed does not imply the properties of the “is-a” relationship. Note that if an instance of the presented structure is accessed through the role *Employee*, the result is not compatible with the *Person* class. The result would then include the *job* attribute being the structure *JobDescription* (e.g. some detailed job description used in a given company) instead of the *String* type value expected for a *Person*. On the other hand, if an object having a role *Employee* is accessed directly (that is, using the “Person” name), calling *calculateTax()* method on it would execute the code defined in the class *Person* (any roles attached to that object would be then ignored).

All those rules need to be explicitly defined as the constraints in the metamodel.

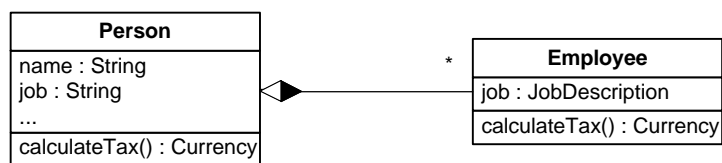


Fig. 5. A role dependency showing the difference between static inheritance related with specialization and dynamic inheritance provided by dynamic roles

Concerning the query language, we found it necessary to introduce or modify the following operators:

- The role-casting operator is necessary to make explicit conversions between a role and its base object, between base object and its role or among different roles of the

same object.⁷ For example (using a popular cast operator syntax) a query selecting all the persons having both *Student* and *Employee* roles could be formulated as follows. First the *Student* roles that coexist in the same tree with *Employee* roles are selected; then, the *Person* objects owning those roles are extracted:⁸

```
(Person) ((Student) Employee)
```

- The “has role” operator, checking if a given object has at least one role of provided name.
- The “roles” operator, returning references to all direct roles of a given object.
- Other important language operators include adding and dropping a role.
- Copying and deleting objects that have roles should be propagated to their roles, to protect referential integrity.

4.4 Other Applications of the Role Construct

Finally, it is desirable to investigate all the possible combinations of newly introduced notion with already present constructs in order to identify possibly useful patterns. Here we can briefly present two interesting applications mentioned in section 2.

One example can be the issue of ontology, that is, the description of local resources required in integration of distributed systems and for mobile agent software. Particularly, the assumed interpretation of local data, which is hardwired into local system’s software needs to become explicit to make the system’s resources meaningful for external parties. Such descriptive elements, referred to as metadata, can be domain-specific (like e.g. the measure used to express the sizes of clothes) or may rather constitute a very general resource description, as it is the case for the Dublin Core Metadata Element Set (DCMES) [4]. This is an example of context-specific information, which may be unnecessary for local applications but critical for others. Moreover, it may be necessary to isolate the existing local applications from the impact of introducing of such descriptions. This would make the object roles an optimum choice. Note also that significant part of such metadata (especially this defined by DCMES) is rather instance-specific (e.g. resource’s creator, source, language) and therefore can be stored with no redundancy within dynamic object roles, which would separate and partition it according to the needs of particular data access context.

Another application of dynamic object roles, located rather outside the conceptual modeling area, is the realization of AOP mechanism within a DBMS environment. Although a DBMS fixes a number of non-functional aspects that remain open in general-purpose programming languages, the aspect paradigm remains relevant here, and dynamic object roles seem to be well suited to provide a base for it. If the repertoire of properties traditionally defined within a class was augmented with the

⁷ Since we allow object to possess more than one role of a given name, the operator would return several role references for such object.

⁸ The operator is assumed to collect results from all levels of role hierarchy (thus accessing also indirect roles).

notion of active rule, the role-classes using this feature may be used as a very flexible mean to implement aspects.

5 Conclusions and Future Work

The dynamic object role is an example of a prominent notion, which is very likely to augment the currently used modeling and implementation languages. Although it bears semblance to several broadly used notions, the role concept is unique and thus requires dedicated solutions to support it in modeling and software development. Since it constitutes an extension of a core data model, a number of different areas have to be considered. Among them are the following:

- conceptual modeling, including a graphical notation;
- consistent and preferably limited change to an original metamodel definition;
- DBMS query language constructs;
- database object store model;
- possible usage of newly introduced concept in modeling and extending different kinds of metadata.

Introduced notion needs to take into account and adjust to pre-existing solutions, however only as far as such compliance does not appear to be a limiting factor. In this paper the intended shape of the considered dynamic object role concept respecting the abovementioned assumptions has been outlined. We suggest a modest extension to DBMS and modeling language metamodels, which introduces a special kind of dependency between the class describing a role and the class describing its player (being another role or a regular object). This dependency (constituting a specific mix or inheritance and composition features) restricts applicability of particular role to appropriate type (class) of its player. Taking into account the overall overhead of introducing the new construct, we intentionally follow a rather lightweight approach to define it.

The prototype of basic DBMS functionality with roles has been developed to verify our object store model proposal and appropriate stack-based approach [14] extensions. Since no typing constraints have been implemented so far, we are now going to experiment with the solutions discussed here. We are also going to further investigate the use of the role mechanism as a base to implement aspects in an ODBMS environment as well as the use of roles to encapsulate metainformation necessary for distributed databases.

References

1. The AspectJ project homepage. <http://aspectj.org>
2. M. Atkinson, R. Morrison. Orthogonally Persistent Object Systems. The VLDB Journal, 4(3), 1995, pp.319-401.
3. R. G. G. Cattell, D. K. Barry: The Object Data Standard: ODMG 3.0. Morgan Kaufmann 2000.
4. Dublin Core Metadata Element Set, Version 1.1. <http://dublincore.org/documents/>

5. M. Fowler. Dealing with Roles. <http://www.martinfowler.com/> (an update to: Analysis Patterns: Reusable Object Models. Addison-Wesley 1996).
6. P. Habela, M. Roantree, K. Subieta: Flattening the Metamodel for Object Databases. Advances in Databases and Information Systems, 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, September 8-11, 2002, Proceedings. Lecture Notes in Computer Science 2435 Springer 2002.
7. A. Jodlowski, P. Habela, J. Plodzien, K. Subieta: Objects and Roles in the Stack-Based Approach. DEXA 2002: 514-523.
8. A. Jodlowski, J. Plodzien, E. Stemposz, K. Subieta: Introducing Dynamic Object Roles into the UML Class Diagram, materialy konferencji IASTED International Conference on Software Engineering and Applications (SEA), ACTA Press, ss. 629-634, Cambridge, MA, USA, 2002.
9. G. Kiczales, J. Lamping, A. Mendhekar, Ch. Maeda, C. Videira Lopes, J.-M. Loingtier, J. Irwin: Aspect-Oriented Programming. ECOOP 1997: 220-242.
10. R. C. Martin. Design Principles and Design Patterns. <http://www.objectmentor.com/>
11. Object Management Group: Meta Object Facility (MOF) Specification. Version 1.4, April 2002 [<http://www.omg.org>].
12. Object Management Group: Unified Modeling Language (UML) Specification. Version 1.4, September 2001 [<http://www.omg.org>].
13. F. Steimann: On the representation of roles in object-oriented and conceptual modelling. DKE 35(1): 83-106 (2000).
14. K. Subieta, C. Beerl, F. Matthes, J. W. Schmidt: A Stack-Based Approach to Query Languages. East/West Database Workshop 1994: 159-180
15. K. Subieta, A. Jodlowski, P. Habela, J. Plodzien: Conceptual Modeling with Dynamic Object Roles. In: "Technologies Supporting Business Solutions" (ed. R. Corchuelo, A. Ruiz-Cortés, R. Wrembel), the ACTP Series, Nova Science Books and Journals, New York, USA.
16. R. K. Wong, H. L. Chau, F. H. Lochovsky. A Data Model and Semantics of Objects with Dynamic Roles. Proc. of Intl. Conf. on Data Engineering, 1997