

Dynamic Object Roles – Adjusting the Notion for Flexible Modeling*

Andrzej Jodłowski
*Institute of Computer Science,
Polish Academy of Sciences,
Warsaw, Poland
andrzejj@ipipan.waw.pl*

Piotr Habela
*Polish-Japanese Institute of
Information Technology,
Warsaw, Poland
habela@pjwstk.edu.pl*

Jacek Płodzień
*Institute of Computer Science,
Polish Academy of Sciences;
Warsaw School of Economics,
Warsaw, Poland
jpl@ipipan.waw.pl*

Kazimierz Subieta
*Institute of Computer Science, Polish Academy of Sciences;
Polish-Japanese Institute of Information Technology, Warsaw, Poland
subieta@ipipan.waw.pl*

Abstract

The conceptual modeling of real-life situations often requires expressing some kind of multiple, multi-aspect or dynamic specialization. On the other hand, multiple or dynamic inheritance impose troublesome anomalies and thus are sporadically and not fully supported by development tools. In this paper we suggest a new approach to the concept of dynamic object roles, capable of expressing all the mentioned special kinds of specialization while avoiding the anomalies of multiple inheritance. The base mechanism of the notion is described and the possible design decisions concerning its realization in a modeling language and implementation, having impact on its overall usability, are discussed.

1. Introduction

Despite a large collection of various conceptual modeling facilities, it is still difficult to model directly and precisely some typical situations in the business reality. An example is the concept of multi-inheritance, which supports conceptual modeling, but leads to various semantic anomalies. Inaccurate modeling results in errors, communication problems for a project's members, additional consumption of resources during system construction, and has negative impact on the code's

length, documentation, transparency, maintainability and reliability. Thus, conceptual modeling should offer all necessary notions that would allow the analyst and designer to express their design vision as precisely as possible.

On the other hand, too many extensions to the existing modeling constructs may cause difficulties concerning the learning curve and proper use by a project's members. Thus there is an opposite tendency to minimize the number of concepts and express new concepts through those already existing. For instance, some methodologies do not deal with aggregation, considering it a special case of association; some do not involve inheritance at all, assuming that it can be expressed otherwise, etc. Another disadvantage of a large number of conceptual modeling notions is their inherent semi-formal semantics (they support humans' thinking rather than computer operation), which could cause difficulties in recognizing a proper usage of the notions and semantics of their particular combinations.

The tendency to extend conceptual modeling notions is also reduced by implementation environments. If some conceptual notion has no direct counterpart on the implementation side, then it must be mapped into other implementation notions; thus the original idea of the analyst/designer is misshapen. In consequence, there is little motivation to use those notions, which have no direct counterparts in implementation. For example, because relational databases do not support inheritance,

* This work was partially supported by the European Commission project ICONS; project no. IST-2001-32429.

corresponding analysis and design methodologies (except for a few) avoid this concept.

1.1. Roles

A promising concept which is believed to be especially useful for complex and dynamic problem domains is dynamic object roles (see e.g. [3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 17, 18, 22, 25, 26]). However, in spite of its potential, it is not supported by popular methodologies, programming languages, etc. One of the main reasons of this low popularity is the already established object-oriented principles, especially in programming languages. The basic assumption is that objects conform to the substitutability principle (LSP), which is natural but leads to anomalies typical for multiple, multiple-aspect and repeating inheritance. Another assumption that impedes the popularity of dynamic roles is strong static (polymorphic) typing, which in the case of roles must be relaxed or redesigned. In this paper we try to convince the reader that the mentioned impediments of a wide usage of roles can be avoided, with minimum impact on the existing modeling notions and methods.

The concept could facilitate such modeling tools as the UML [16] and could be an important paradigm for object-oriented databases constructed e.g. in the spirit of the ODMG [15]. The notion is already present (under another name and with specific semantics) in the standard SQL3 (abandoned) and its successor SQL1999 [2].

The rest of the paper is structured as follows. Section 2 indicates problems of multiple inheritance. Section 3 describes our object model with roles. Section 4 looks for an optimum match of the notion's features with the requirements of modeling and subsequent implementation. Section 5 presents our conclusions and future plans.

2. Problems with multiple inheritance

Because of the great number of relevant material, the issue of multiple inheritance is only outlined in this section. Multiple inheritance is useful during the conceptual modeling stage, but because it is often not supported by implementation tools it cannot be applied during further phases of software development. The concept is absent in some object-oriented programming languages (e.g. Smalltalk, Java¹) and database standards

(e.g. ODMG²), or it leads to anomalies (e.g. C++). Thus, designers and programmers may have difficulties to map the conceptual design into implementation decisions. The problem is amplified during the maintenance phase, where many particular implementation decisions avoiding multiple inheritance must be reversely mapped into the conceptual model.

Most of the object-oriented analysis and design methodologies and notations introduce multiple inheritance as a basic feature, e.g. the OMT [20] and the UML. Their authors argue that the absence of multiple inheritance in programming tools results in a disadvantageous gap between a conceptual model and its implementation model. Unfortunately, for these artifacts the semantics of multiple inheritance is not sufficiently precise. It is usually recommended to avoid name conflicts through unique method naming in all classes or to avoid multiple inheritance by class permutations or delegation.

The problem of multiple inheritance concerns the inheritance of properties with the same name from different classes. The tradeoff methods are often seen by opponents of multiple inheritance as examples of pathologies that occur in the understanding of this concept. For instance, class priority is usually defined through syntactic ordering of super classes, which are then used to define specialized classes through multiple inheritance. Some systems avoid name conflicts through defining the sequence of classes reflecting their priority. For example, if classes A and B have methods with the same name *m*, and class C inherits both from A and B, then a call (message) *m* means invocation of *m* defined in A for all members of C. The priority rule violates substitutability, because C objects cannot be used in such contexts where B objects are used and *m* is invoked. This approach obviously undermines polymorphism.

Usually however, languages supporting multiple inheritance do not possess built-in priority rules for multiple inheritance. Thus the only choice is to avoid name conflicts. This in turn violates the *open-closed principle*³, a base for class reuse. It may be supported by additional mechanisms in a language, during the construction of code (e.g. in C++) or by a proper construction of classes (e.g. in Eiffel). The detailed methods of solving name conflicts are:

¹ Note that Java provides multiple inheritance of interfaces, but not classes, which prohibits inheriting structural properties and implementation. Thus this kind of inheritance is severely limited.

² "Due to the inefficiencies and ambiguities of multiple inheritance of state" [15] the ODMG Object Model supports multiple inheritance of interfaces, but not classes.

³ The principle says that classes are closed for changes but open for specializations. If classes *A* and *B* are developed independently and then closed, a name conflict causes that specialization *C* inheriting both from *A* and *B* is impossible.

- Local renaming (change) of an inherited property in a given subclass – O2, Eiffel (violates the open-closed principle).
- Limitation of the scope for inherited properties – a selection mechanism for the inheritance in Eiffel.
- Specification (for a name conflict) of a class qualifier in all (or some) places, where inherited properties are used – C++ with scope operator “::” (violates the open-closed and substitutability principles).
- The automatic or semiautomatic renaming of attributes or methods, which results in overheads during program development and maintenance, and is error-prone.

Typically, a class structure is unchangeable during the whole (or at least a part of the) life of a system, especially if it is implemented within a database schema. Most often, a class structure cannot be modified during program execution, even if classes have the first-class citizenship. This means that independently of an option of resolving name conflicts, one or both of the mentioned object-oriented principles is violated, with consequences to the productivity of program developers, and understandability, reliability and maintainability of applications.

3. Dynamic role concept

The idea of dynamic object roles is simple and natural. It assumes that every real or abstract entity during its life can acquire and lose many (an arbitrary number of) roles without changing its identity. Roles appear during the life of a given object, they can exist simultaneously, and they can disappear at any moment. For example, a certain person can at the same time be a student, an employee, a patient, a club member, etc.

Roles cannot exist without their owner entities (also called role players or role base objects), e.g., a *Patient* role cannot exist without its *Person* entity. Deleting an object causes deleting all of its roles. Roles can exist simultaneously and independently of each other. Roles are treated as a special kind of objects, they can possess their own roles as well, which allows one to create subroles, sub-subroles, etc. A role can have its own additional attributes and methods. Two roles can contain attributes and methods with the same names, but this does not lead to conflict. Also, as in the case of regular objects, classes describing roles can be specialized. For example, the specialization of a role *Club_Member* can be a role *Club_President*. This is a fundamental difference in comparison to multiple inheritance.

Relationships (associations) between objects can also connect objects with roles and roles with roles. For example, a relationship *works_in* could connect an *Employee* role with a *Company* object. This makes the

referential semantics clean in comparison to the traditional object models, where relationships connect entire objects, and not their specialized parts.

Dynamic object roles are especially useful for temporal databases as roles can represent any past facts concerning objects, e.g. an employment history through many *Employee* roles within one *Person* object.

In our approach we introduce roles through composites of objects with a special structure and semantics. A version of roles has been implemented in the prototype system Loqis [23]. Currently we are working on a prototype of an object-oriented DBMS to be used for content management for Web applications, where we intend to implement the ideas presented in this paper. In particular, we have developed a query/programming language for it in the spirit of ODMG OQL, with its semantics based on the stack-based approach [19, 24]. A more thorough discussion of the dynamic object role mechanism considered for this environment can be found in [8].

3.1. The main features of dynamic roles

In our approach we assume that an object can contain many sub-objects⁴ called roles. These subobjects can be inserted and removed at run time, as in [1, 21]. Roles have different types and can exist simultaneously and independently. A role has its own attributes and behavior. Identical names in two or more roles of different types do not imply any semantic dependency between corresponding properties. For example, a person can play simultaneously the role of an employee of a research institute with an attribute *Salary*, and the role of an employee of a service company with another attribute *Salary*. These two attributes exist at the same time, but except for the name no other feature is shared, including types, semantics and business ontologies. A role dynamically “imports” attributes (values) and behavior from its super-roles, in particular, from its parent object.

In Figure 1 we present an example showing basic features of our store model with dynamic roles. The following features are shown:

- An object (shown as a rectangle with attribute values), e.g. *Person*, may have any number of specialized roles (e.g. *Employee* and *Student*).
- Each role has its own name which can be used to bind the role from a program or a query. The presented objects can be bound through name *Person* (each), through name *Employee* (1st and 2nd) and

⁴ It does not necessarily mean the physical nesting of roles in their base objects. This metaphor is only intended to emphasize the fact that each role [instance] belongs to exactly one base object and unless reassigned, cannot outlive it.

through name *Student* (2nd and 3rd). Each binding returns the identifier of a proper role (or the identifiers of proper roles for multi-valued bindings).

- Each role is encapsulated, i.e. its properties are not seen from other roles unless it is explicitly stated by a special link (shown as a line with a black diamond-white triangle combination symbol). In particular, a role *Employee* imports all properties of its base object *Person*. For example, if the first object is bound by name *Person*, then the properties {*Name* “*Brown*”, *BirthYear* 1975} are available; however if the same object is bound by name *Employee*, then the properties {*Salary* 2500, *Job* “*analyst*”, *Name* “*Brown*”, *BirthYear* 1975} are available.

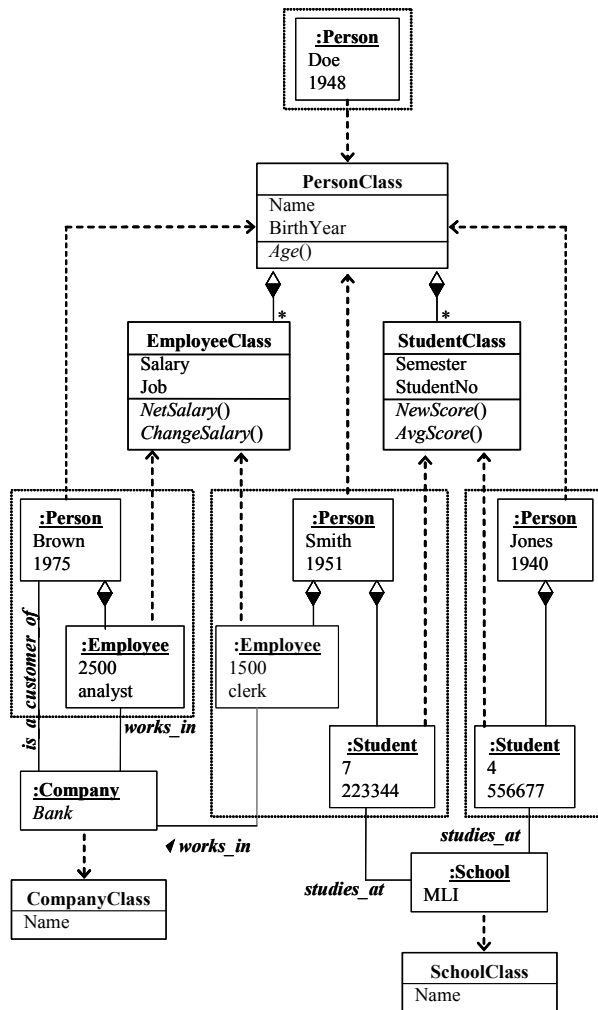


Fig. 1. Objects, roles and classes

- Each role/object is connected to its own class. The connection is shown as a thick dashed arrow, leading from an instance to its class. Classes contain invariant properties of corresponding roles, in

particular, names (the first section), attributes and their types (the second section; attribute types are not shown) and methods (the third section).

- Links can join not only objects with objects (as it is possible in the “traditional” approaches), but also objects with roles and roles with roles. For example, in Figure 1 a link *works_in* joins an object *Company* with a role *Employee*. Indirectly, this link leads to *Person*, because the role *Employee* imports the properties of its parent object *Person*. However, after accessing the object via such a link, the properties of the other object’s roles (here: *Student*) remain invisible.
- Although not considered in the core object store model, the relationships between role classes and their intended base objects’ classes can be specified to describe applicability of particular roles (see e.g. symbol between *EmployeeClass* and *PersonClass* in Figure 1). This issue, very important for modeling and data consistency checking, will be further discussed in this paper.

3.2. A store model with roles

For simplicity and for making the semantics clean, we assume object relativism (i.e. each property of an object is an object too) and consider all properties of the store (including classes) being first-class citizens. The store models a program/database state does not involve types, which we consider a checking utility rather than a materialized property of the state.

In our approach an object is a triple $\langle i, n, v \rangle$, where i is a unique internal object’s identifier, n is an external object’s name, and v is an object’s value. Such an object can be an atomic one (if its value is atomic, e.g. “Doe”), can be a reference one (if its value is a reference to another object), or can be a complex one (if its value is a set of objects). Classes and methods constitute special kinds of objects. The object store model is presented as a 6-tuple $\langle O, C, R, CC, OC, OO \rangle$, where:

- O is a collection of objects,
- C is a collection of classes,
- R is a collection of root identifiers (that is, identifiers of objects being entries to the store),
- CC is a binary relation determining inheritance relationship,
- OC is a binary relation determining membership of objects within classes,
- OO is a binary relation determining inheritance relationship between roles.

The tuple does not contain an element for links between objects, because links are represented as reference objects stored within complex objects.

In Figure 2 we present a simple example of an object store built according to the definition. We must now distinguish objects and roles. Objects may consist of many roles but a role belongs to a single object. A base object can be considered a main role with name reflecting the semantics of the entire object; deleting it implies deletion of the entire object. Any role can inherit dynamically from another role within the same object; inheritance among roles in different objects is forbidden. This kind of inheritance is known from prototype-based languages, e.g. Self. The relation OO defines two functional aspects. First of all, it determines which roles are inherited by other roles. On the other hand, it fixes the semantics of manipulating objects with roles. In particular, copying an object implies “isomorphic” copying of all of its roles, and deleting an object implies deletion of all its roles. Deleting a role implies recursive deletion of all its sub-roles. OO is a pure hierarchy (that is, there are no cycles). Each role directly inherits from its class and through super-roles indirectly inherits the properties of the classes that these super-roles belong to.

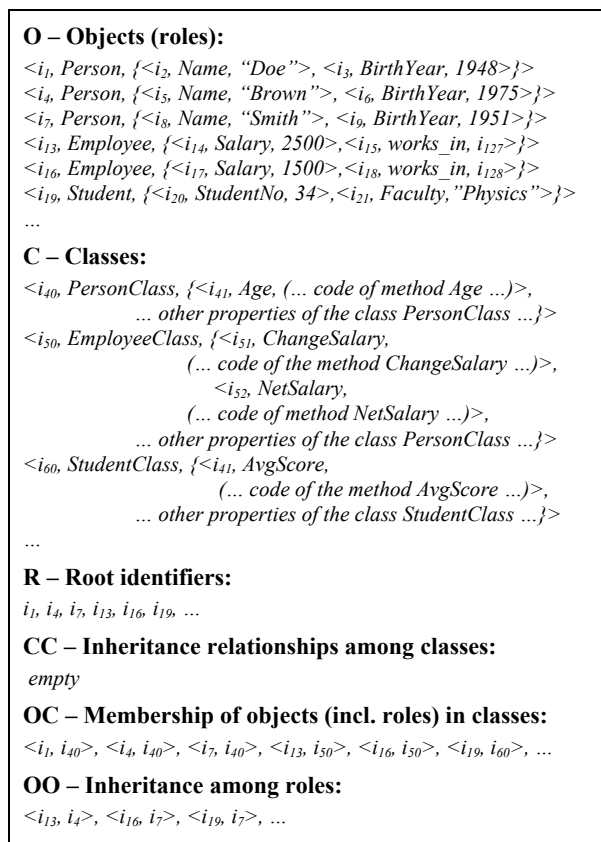


Fig. 2. An exemplary state of the object store with roles

4. Modeling and implementation: tuning the role notion

While we assume the abovementioned object-level role-related mechanisms as capable of overcoming the static inheritance problems, the exact treatment of the notion in modeling and in schema management can significantly affect the overall usability of the object model with roles. In this section we present various requirements concerning dynamic roles’ features and try to provide an optimum solution.

4.1. Dynamic role – possible usage patterns

The concept of role is very frequently used in different contexts of the real world. Analogously, its possible applications in modeling of information systems may serve very different purposes and generate various (perhaps contradictory) requirements concerning the notion’s exact shape. Three specific flavors of the role concept interpretation are depicted in Figure 3 using an ad-hoc notation extending the UML.

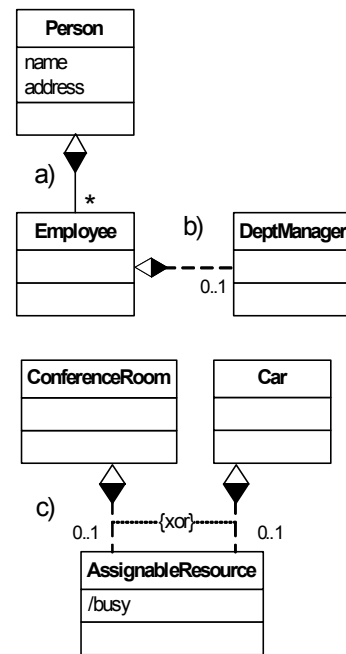


Fig. 3. Examples of different kinds of role dependency, introducing exemplary notation to distinguish them

The least restrictive approach to the role usage would impose no additional constraints apart from the ones resulting from the assumed object store model. This would mean the ability to connect any two objects with the “is_a_role_of” relationship as long as the resulting

structure remains a hierarchy. Such a solution offers maximum flexibility, however, it does not prevent inconsistencies (i.e. attaching the role to an object that is not intended to possess a role of a given kind) and errors (i.e. absence of the main object's properties required by a given role's behavior).

The latter of the above-mentioned problems is not always the case, since some roles do not have to make assumptions on their base objects' features. They may serve as a kind of flag used to group different objects or may provide some additional annotation. Case "c" in Figure 3 shows an example of an *Assignable Resource* role that may be played by either *Car* or *Conference Room* objects. Note that even if there is no dependency on any base object's property, it still may be practical to indicate the intended kinds of base objects (as shown in the figure).

Another case would be a "vacant" role, that is, a role object that is allowed to exist independently of any base objects in some phases of its lifecycle. For example (Figure 3, case "b"), some business tasks would be assigned to the *Dept Manager* role even if it was temporarily not played by any of employees. However, this pattern of the role concept usage will not be further discussed, as we considered the required additional features (i.e. checking the presence of a base object) too costly in terms of overall complexity to justify them.

Finally, case "a" in Figure 3 shows the role dependency that seems to be the most conventional due to its similarity with plain, static inheritance. Here, a role is required to always coexist with its base object, and the exact class of this base object is specified. With such constraints, the methods (not shown in the figure) defined within the *Employee* role, can safely refer to any property defined by the *Person* class. Also external code may access the *Person* class-defined properties when dealing with *Employee*. Note however, that at the same time, it is the most restrictive of the discussed approaches to modeling roles.

4.2. Flexibility vs. consistency

The overview presented in the previous subsection indicates two contradictory needs concerning the allowed role usage. On the one hand, the specification of the class of a role's base object supports consistency and provides a foundation for enforcing type safety. This style of definition would also be easier to accept for developers who are used to applying mainstream object model features. For those reasons, we chose this solution [9] as the primary approach to dynamic roles in our prototype ODBMS currently under development.

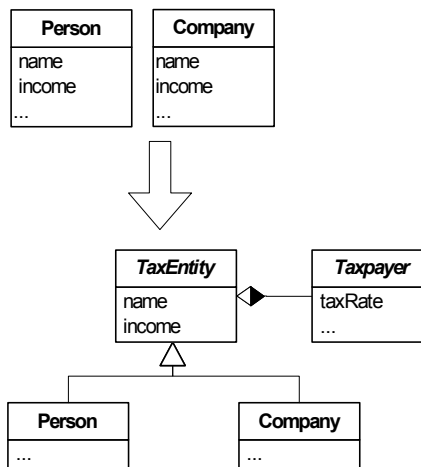


Fig. 4. An example of model transformation needed to make a *Taxpayer* role assignable to both the *Person* and *Company* instances

On the other hand, there are situations where the requirement of specifying a single base object's class constitutes an obstacle for effective modeling. If there are two or more classes of base objects to which a given role is applicable, there are two ways of keeping the model consistent:

- If there is no specific base object's properties used by a given role, the modeler can associate it with a root of class hierarchy (usually called *Object*) as its base object class. The only problem of this solution is blurring the original intent, since the declared applicability of the role is broader than necessary.
- If there are some properties of base objects required by a role, they need to be grouped in the common superclass of the designated base objects classes (see the example in Figure 4).

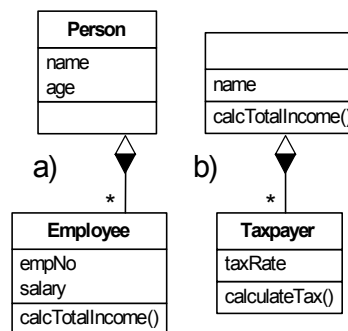


Fig. 5. Two kinds of role's player specification: a role class assigned to specific base class (a) and a role class whose base object's properties required are specified through an import list (b)

Although it concerns very specific cases, note that the second approach is not universally applicable. The common superclass that is required may be perceived unnatural. Moreover, under some circumstances it may even require multiple inheritance, what we are trying to avoid.

To overcome this problem without giving up the role applicability constraints, we may suggest introducing an additional way of declaring the required base objects' features. This may take the form of "import specifications", where a role applicable to various object types would specify the features required (attributes, relationships and operations), without the need of indicating a specific class they belong to.

Fig. 5 illustrates this approach and shows the difference to the solution discussed earlier. In contrast to the case a), where the role *Employee* shows its applicability to specific base class, in case b) only the features needed by a role *Taxpayer* are specified. Those features need to be specified in a data definition language with similar details as regular class's properties; in the visual notation we used, they are grouped within an anonymous class. This is intended to state that any object having the attribute *name* and operation *calcTotalIncome()*⁵ can safely play the role *Taxpayer*. This may be assumed as a move towards structural type match applied for interdependencies between roles and the types of their base objects.

5. Conclusions

In the paper we have presented a concept of dynamic roles, which can extend object-oriented languages and database object models, such as the ODMG object model. Dynamic roles can support conceptual models of many applications and, in comparison to classical object models, do not lead to anomalies and limitations of multiple inheritance. We also consider dynamic roles an alternative to other problematic constructions such as repeating and multi-aspect inheritance. The problems with the object migration issue can also be avoided by dynamic roles. We argue that the concept extends the expressiveness of conceptual modeling and can be consistently introduced into the implementation level.

Classical concepts of object-orientation, such as object identity, polymorphism and overriding can be smoothly incorporated into the model. An advantage of the model with dynamic roles is conceptual clarity concerning the level of object store and the mechanisms of data naming, scope control and binding names.

⁵ Although not shown in the diagram, we of course assume specifying attributes' types and methods' signatures in both modeling and data definition language syntax.

As a further step towards a consistent view of the object model with dynamic roles, we have presented the different cases of the usage of the notion and a possible way of fitting it into traditional object model. To resolve the consistency checking vs. flexibility dilemma, an additional construct to be used in a role specification has been suggested.

The model with dynamic roles, a novel query/programming language, and data definition language are currently being developed as a part of our ODBMS prototype project.

6. References

- [1] A. Albano, R. Bergamini, G. Ghelli, R. Orsini, "An Object Data Model with Roles", *Proc. of the VLDB Conf. 1993*, pp. 39-51.
- [2] American National Standards Institute (ANSI) Database Committee (X3H2), *Database Language SQL Part 2: Foundation (SQL/Foundation)*, J. Melton, Editor, Working Draft, March 1999.
- [3] E. Bertino, G. Guerrini, "Objects with Multiple Most Specific Classes", *Proc. of the ECOOP Conf. 1995*, Springer LNCS 952, pp. 102-126.
- [4] D. Bäumer, D. Riehle, W. Siberski, M. Wulf. "Role Object", *Pattern Languages of Program Design 4*, Edited by N. Harrison, B. Foote, H. Rohnert. Addison-Wesley, 2000, pp. 15-32.
- [5] M. Fowler, *Dealing with Roles*, 1998, <http://www2.awl.com/cseng/titles/0-201-89542-0/roles2-1.html>
- [6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [7] G. Gottlob, M. Schrefl, B. Rock, "Extending Object-Oriented Systems with Roles", *ACM Transactions on Information Systems*, 1996, pp. 268-296.
- [8] A. Jodlowski, P. Habela, J. Plodzien, K. Subieta, "Objects and Roles in the Stack-Based Approach", *Proc. of the DEXA Conf. 2002*, Springer LNCS 2453, pp. 514-523.
- [9] A. Jodlowski, P. Habela, J. Plodzien, K. Subieta, "Extending OO Metamodels Towards Dynamic Object Roles", *Proc. of the ODBASE Conf. 2003*, Springer LNCS 2888, pp. 1032-1047.
- [10] G. Kappel, S. Rausch-Schott, W. Retschitzegger, "A Framework for Workflow Management Systems Based on Objects, Rules and Roles", *ACM Computing Surveys* 32, 2000, p. 27.

- [11] B.B. Kristensen, K. Østerbye, "Roles: Conceptual Abstraction Theory and Practical Language Issues", *Theory and Practice of Object Systems* 2(3), 1996 pp. 143-160.
- [12] B.B. Kristensen, "Object-Oriented Modeling with Roles", *Proc. of the OOIS Conf. 1995*, pp. 57-71.
- [13] Q. Li, F.H. Lochovsky, "ADOME: An Advanced Object Modeling Environment", *IEEE Transactions on Knowledge and Data Engineering* 10(2), 1998, pp. 255-276.
- [14] Q. Li, R.K. Wong, "Multifaceted Object Modeling with Roles: A Comprehensive Approach", *Information Sciences* 117(3-4), 1999, pp. 243-266.
- [15] Object Data Management Group, *The Object Database Standard ODMG, Release 3.0*, R.G.G. Cattell, D.K. Barry, Eds., Morgan Kaufmann, 2000.
- [16] Object Management Group: *OMG Unified Modeling Language Specification, Version 1.5*, The Object Management Group, March 2003, <http://www.omg.org>
- [17] M. Papazoglou, "Roles: A Methodology for Representing Multifaced Objects", *Proc. of the DEXA Conf. 1991*, pp. 7-12.
- [18] B. Pernici, "Objects with Roles", *Proc. of the IEEE/ACM Conf. on Office Information Systems, 1990*.
- [19] J. Plodzien, A. Kraken, "Object Query Optimization through Detecting Independent Subqueries", *Information Systems* 25(8), 2000 pp. 467-490.
- [20] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Englewood Cliffs, New Jersey, Prentice Hall International, Inc., 1991.
- [21] J. Richardson, P. Schwarz. "Aspects: Extending Objects to Support Multiple, Independent Roles", *Proc. of the ACM SIGMOD Conf. 1991*, pp. 298-307.
- [22] D. Riehle, T. Gross, "Role Model Based Framework Design and Integration", *Proc. of the OOPSLA Conf. 1998*, ACM Press, pp. 117-133.
- [23] K. Subieta, F. Matthes, J.W. Schmidt, A. Rudloff, I. Wetzal, "Viewers: A Data-World Analogue of Procedure Calls", *Proc. of the VLDB Conf. 1993*, pp. 269-277.
- [24] K. Subieta, Y. Kambayashi, J. Leszczykowski, "Procedures in Object-Oriented Query Languages", *Proc. of the VLDB Conf. 1995*, pp.182-193.
- [25] R.K. Wong, H.L. Chau, F.H. Lochovsky. "A Data Model and Semantics of Objects with Dynamic Roles", *Proc. of the ICDE Conf. 1997*, IEEE Computer Society, pp. 402-411.
- [26] R.K. Wong, Q. Li, "Manufacturing Systems Modeling with Roles. A Comprehensive Approach", *Proc. Of the Sixth IFIP TC-2 Working Conference on Data Semantics (DS-6) 1995*, pp. 461-478.