A Comparison of Role Mechanisms in Object-Oriented Modeling

Gerti Kappel, Werner Retschitzegger, Wieland Schwinger

Department of Information Systems
Johannes Kepler University of Linz, Austria
{gerti | werner | wieland}@ifs.uni-linz.ac.at

Abstract. In most object-oriented languages and systems, objects must belong to a single most specific class. Relaxing this requirement, also known as role modeling, has been proposed by several research groups independently. To shed some light on the similarities and differences of the proposed solutions we provide an overview and comparison thereof in this paper.

1. Introduction

Object-oriented modeling and development has left research laboratories and is widely used in industry. Not least since Aksit's and Bergmans' paper at OOPSLA'92 [Aksi92] several problems, however, with object-oriented modeling are known and solutions thereof are sought. One of the problems, the lack of adequate object evolution mechanisms - also known as role modeling - has attracted the attention of several research groups. Meanwhile different proposals for extending the traditional object model with role mechanisms have been published. To be able to reason about the different approaches an overview and comparison of their several features is badly needed. This paper is a first step in this direction. In the next section, we sketch the problem of object evolution. Section 3 introduces five approaches to cope with this problem and provides a comparison of the proposed solutions. Section 4 concludes the paper.

2. The Problem

In traditional object-oriented models an object may be instance of exactly one class (class in the sense of type, i.e., the specification of structure and behavior). It is in general not possible that an object changes its class membership during its lifetime, and that it is instance of several classes at the same time. This is a major obstacle in the expressive power of object-oriented models. Firstly, if the behavior of an object changes, the object must be "re-created" as new instance of the class representing the newly required behavior, values of the old instance must be copied to the new instance, references must be redirected, and finally the old instance has to be deleted. It is obvious that this task is both error-prone and tedious. Secondly, if an object exhibits context-dependent behavior, the union of all behaviors of all contexts has to be represented within one class, of which the object is an instance. Such a "union class" may be modeled via multiple inheritance, if the latter is supported in the underlying object model. However, multiple inheritance both incorporates the naming problem and exhibits all context-dependent behavior at the same time. And thirdly, it is not possible that an object gets some basic general-purpose behavior on the occasion of its birth, and learns and forgets specific behavior depending on the context in which it is currently used throughout its lifetime. This context-dependent behavior is also known as *role behavior*, since it is commonly accepted nomenclature that an object with changing behavior is *playing one or several roles* and that it may *learn and forget roles*. (Note, this notion of a role is in contrast to roles in semantic data models, such as the ER model and the NIAM model, where a role models the part of an entity within a specific relationship [Bron95].) Encountering this obstacle the traditional object-oriented model has been extended to cover role-related behavior. Some of these extensions are introduced in the next section.

3. Proposed Solutions

The notion of role in data modeling is almost as old as data modeling itself and has been introduced by Bachman and Daya [Bach77]. Later on, the restriction that an object may be associated with a single class only has been relaxed in the object-oriented database system Iris [Fish87] and the object-oriented knowledge based system MOKUM [Riet89]. The importance of context-dependent behavior modeling has been re-discovered with the advent of object-oriented systems and languages in the large. In all commonly used object-oriented languages and database systems an object is exclusively connected to a single class only and may not change this class membership throughout its lifetime. This has been the motivation for several proposals on incorporating the role concept into the "pure" object-oriented model. In the following, we will have a closer look at some of these proposals, namely, Clover [Stei89], Aspects [Rich91], Fibonacci [Alba95], OBD and extended Smalltalk [Kapp91, Gott96], and DOOR [Wong97].

There are some other proposals on role modeling in literature, most notably by Wieringa et al. [Wier91, Wier94], Bertino et al. [Bert95], Pernici [Pern90], Elmasri et al. [Elma90, Elma92], and Sciore [Scio89], which we do not deal with in detail in this paper. Wieringa et al. and Bertino et al. put forward models and algorithms, respectively, on how to unambiguously dispatch an attribute and a method on an object in the presence of multiple most specific classes. They do not further deal with role modeling issues such as role migration and role suspension. Pernici has been one of the very first on providing a constraint-based language for specifying inter-role dependencies. A role in her model, however, is "only" a part in an object's life cycle specifying these states and state transitions which together represent a meaningful period or view on the object's lifespan. Again, issues such as role migration and viewing an object in a specific role have not been dealt with. Elmasri et al. have defined a temporal data model based on an extended Entity/Relationship model with entity role types. An entity role represents the time that the entity is participating in that role. Though related, Elmasri's work is not in the realm of object-oriented modeling, and thus not further considered in this paper. Sciore has put forward the notion of object hierarchies for modeling real world entities. If a message is sent to an object, the object may either directly respond to the message, or delegate the message to its parent object. Thus, Sciore's approach is similar to the prototype-based approaches rather than to the class-based ones. Due to the absence of the notion of a class, class-related modeling such as inheritance hierarchies and strong typing is not supported.

Last but not least, it is interesting to note that already semantic data modeling methods [Smit77] as well as object-oriented modeling methods, such as OMT [Rumb91] and UML [UML97], do not require that an object is associated with a single most specific class, either. Thus, with these modeling methods it is possible to represent overlapping inheritance in the sense that an object is instance of several classes at the same time, the parent class and two or more child classes. However, these models fall short on modeling role-specific behavior, such as role migration, and role suspension.

In the following a closer look at the above mentioned role modeling proposals is given.

Clover: Stein and Zdonik have proposed a mechanism called *clovers* [Stei89] in the realm of traditional subtyping and inheritance. An object is created as instance of a specific type, and may become, i.e., may be extended to an instance of one or several subtypes of this type. An object having several such leaf-like extensions looks like a clover, thus the name of this mechanism. The operator for extending an object is called `become`, the operators for viewing an object in its roles as a supertype and a subtype, respectively, are called `coerce-` and `coerce+`. Since clovers are defined in a strongly typed setting type safety is ensured. To ensure type safety also with the `coerce+` operator, a typechecking guardian is associated with it. An object with its several leafs has different identifiers.

Aspects: Richardson and Schwarz propose a strongly-typed model based on interfaces (types) and implementations. Their objects may have multiple *aspects* in the sense of roles and may be extended with new ones during their life time [Rich91]. An object is always seen via one of its aspects. Interfaces are defined separately from implementations. The interface matching is structural, thus implicit conformity, but no inheritance is supported, neither on the aspect level nor on the type level. Since aspects do not support inheritance, an aspect extending an object must explicitly replicate the object's interface in its definition. It may call the original methods via a send-to-super primitive. An object together with its aspects exhibits a single object identifier only. Two comparison operators are supported, one for references to the same aspect of the same object, and one for references to any two aspects of the same object.

Fibonacci: Albano et al. have defined a strongly typed object- oriented database programming language, where an object simply consists of an identity and of an acyclic graph of roles [Alba95]. There exist two type hierarchies, an object type hierarchy and a role type hierarchy. A role type hierarchy, also called an object role family, is always rooted at an object type. A role type hierarchy may be dynamically extended by defining a new role type as subtype of one or several existing role types. Ambiguous method dispatch in the presence of multiple inheritance, or in the presence of an object with multiple roles is avoided, due to the following two rules: (i) the most specific behavior prevails, (ii) the most recently acquired behavior prevails. In addition to this late binding approach, a strict binding approach is also supported where a role evaluates a method without considering the possible redefinitions of the method in its subroles. Each role can be dynamically added. A comparison operator allows to test if two roles refer to the same object. In addition, role inspection, i.e., to test if an object has a certain role, and role coercion, i.e., to coerce an object to one of its possible roles, is supported. An object in its several roles has different identifiers.

OBD and extended Smalltalk: Kappel and Schrefl have developed an object-oriented modeling notation called Object/Behavior Diagrams [Kapp91], which explicitly supports the modeling of roles next to objects. Object types are arranged in an object type hierarchy. Role types modeling the context-dependent behavior of an object type are arranged in a role hierarchy rooted at this object type. In addition, role types are "normal" object types and may themselves be organized in some object type hierarchy. Thus, objects and roles have their individual identifiers, and some role is related to its superordinate role or its object, respectively, via an implicitly modeled instance variable `roleOf`. Method dispatch is done by looking for an implementation in the type hierarchy of the receiving object, and if no implementation is found, the type hierarchies of the `roleOf` related objects are recursively inspected, until the root object is reached. Gottlob et al. have extended Smalltalk by adding a few classes to support the role mechanism similar to OBD [Gott96]. The nice feature here is that there is no need to modify the semantics of Smalltalk itself. The most important classes are `ObjectWithRoles`, `RoleType`, and `QualifiedRoleType`. With a qualified role type it is possible to model that a real world entity may have several occurrences of that role type. To be able to distinguish the different occurrences of a role type of a single object, the occurrences have to be uniquely identified via a qualifying attribute. Each role can be dynamically added and abandoned. A comparison operator allows to test if two roles refer to the same object. In addition, role inspection, i.e., to test if an object has a certain role, and role coercion, i.e., to coerce an object to one of its possible roles, is supported.

DOOR: Wong et al. have proposed a Dynamic Object-Oriented database programming language with roles [Wong97]. The salient features of DOOR with respect to roles are multiple role instantiation, role migration between different objects, and seven different method lookup schemes. Multiple role instances of the same role type of the same object may be distinguished by a boolean expression. In case of multiple roles satisfying the expression, the system will only return one of them for the sake of simplicity and efficiency. Since roles and their corresponding superordinate objects have different identifiers, it is possible to migrate a role from one object to another. The semantics of method lookup is based on the idea of delegation known from prototype-based classless languages. The several method lookup schemes are implemented as meta object protocols so that users can change the different method lookup schemes for different specific applications.

Table 1 provides a summary of the discussed features of the different role modeling proposals.

Approaches ***Evaluation Criteria***	Clovers	Aspects	Fibonacci	Extended Smalltalk	DOOR
Underlying Model	own new language	Melampus, based on Emerald	own new language, based on Galileo	basic Smalltalk	own new language
Type Safety	✓	✓	✓	✓	?
Class/Type Hierarchy	✓	✓	✓	✓	✓
Role Hierarchy	✘	✘	✓	✓	✓
Dynamic Creation of Role Types / of Roles	✓✓	✓✓	✓✓	✓✓	✓✓
Role Inspection	✓	✘	✓	✓	✘
Role Coercion	✓	✘	✓	✓	✘
Multiple Role Instantiation	✘	✘	✘	✓ (with unique qualification)	✓ (with no unique qualification)
Role Migration	✘	✘	✘	✘	✓
Method Dispatch	based on type hierarchy; no late binding	no method dispatch for aspects; methods have to be reimplemented for aspects	based on a role & type hierarchy down and up the hierarchy; late binding	based on a role & type hierarchy only up the hierarchy; late binding	based on a role & type hierarchy down and up the hierarchy; 7 method dispatch schemes are supported; late binding?
Identity Semantics	an object with its leafs has different id's	an object and its aspects have a single id	an object in its roles has different id's	objects and roles have different id's	objects and roles have different id's
Check on Role Identity / Object Identity	✓✓	✓✓	✓✓	✓✓	✘✘

Legend: ✓ .... supported, ✘ .... not supported, ? .... unknown, id .... identifier

Table 1: Comparison of the different role modeling proposals

4. Outlook

All work discussed in this paper has its background in semantic data modeling and databases. However, there is a growing awareness of the problem of object evolution in the object-oriented programming language community [Mätz97]. Extensions like subject-oriented programming [Harr93], composition filters [Aksi93], and aspect-oriented programming [Kitz96] are but a few examples of these developments. That an object should exhibit different views representing different behavior of that object has not been newly invented by CORBA and Java, where views are called interfaces, but has already been proposed by Shilling and Sweeney [Shil89]. Moreover, component-oriented technology [Nier95] and design patterns for change [Gamm94] provide yet other mechanisms for coping with the need to flexibly adapt objects to unforeseen requirements. Currently, we are extending our comparison in these directions.

References

[Aksi92] Aksit M., Bergmans L.: "Obstacles in Object-Oriented Software Development", Proc. of the Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92), Vancouver, British Columbia, Canada, Special Issue of SIGPLAN Notices, ed. Andreas Paepcke, Vol. 27, Oct. 1992, pp. 341 - 358.

[Aksi93] Aksit M., Wakita K., Rosch J., Bergmans L., Yonezawa A.: “Abstracting object-interactions using composition-filters”, Object-Based Distributed Programming, eds. Guerraroui R, Nierstrasz O., Riveill M., Springer, LNCS 791.

[Alba95] Albano A., Ghelli G., Orsini R.: “Fibonacci: A programming language of object databases”, VLDB Journal, 4(3):403-444, 1995.

[Bach77] Bachman C. W., Daya M.: “The Role Concept in Data Models”, Intl. Conf. on Very Large Databases, 1977, pp. 464 - 476.

[Bert95] Bertino E., Guerrini G.: “Objects with Multiple Most Specific Classes”, Proc. of the 9th European Conf. on Object-Oriented Programming, ECOOP’95, ed. Olthoff W., Springer, 1995.

[Bron95] Bronts G. H. W. M., Brouwer S. J., Martens C. L. J., Proper H. A.: “A Unifying Object Role Modelling Theory”, Information Systems, Vol. 20, No. 3, 1995, pp. 213 - 235.

[Elma90] Elmasri R., El-Assal I., Kouramajian V.: “Semantics of Temporal Data in an Extended ER Model”, Proc. of the 9th Intl. Conf. on the Entity-Relationship Approach, Elsevier Science Publisher (North-Holland), October 1990.

[Elma92] Elmasri R., Kouramajian V.: “A Temporal Query Language Based on Conceptual Entities and Roles”, Proc. of the 11th Intl. Conf. on the Entity-Relationship Approach, October 1992, pp. 375 - 388.

[Fish87] Fishman et al: “Iris: An object-oriented database management system”, ACM Trans. on Office Information Systems, 5(1):48-69, January 1987.

[Gamm94] Gamma E., Helm R., Johnson R., Vlissides J.: “Design Patterns - Elements of Reusable Object-Oriented Software”, Addison-Wesley Professional Computing Series, 1994

[Gott96] Gottlob G., Schrefl M., Röck B.: “Extending object-oriented systems with roles”, ACM Trans. on Information Systems, Vol. 14, No. 3, July 1996.

[Harr93] Harrison W., Ossher H.: “Subject-Oriented Programming (A Critique of Pure Objects)”, OOPSLA’93, Conference Proceeding, ACM Press, Oct. 1993.

[Kapp91] Kappel G., Schrefl M.: “Object/Behavior Diagrams”, Proc. of the 7th Intl. Conf. on Data Engineering, IEEE Computer Society Press, Kobe, Japan, 1991.

[Kitz96] Kiczales G., des Rivires J., Bobrow D. G.: “Beyond the black box: Open implementation”, IEEE Software, Vol. 13, No. 1, 1996.

[Mätz97] Mätzl K.-U., Bischofberger R. W.: “Designing Object Systems for Evolution”, Theory and Practice of Object Systems, Vol. 3(4), 1997, pp. 265 - 283.

[Nier95] Nierstrasz O., Tsichritzis D. (eds.): “Object-Oriented Software Composition”, Prentice-Hall Object-Oriented Series, Prentice Hall, 1995.

[Pern90] Pernici B.: “Objects with roles”, IEEE/ACM Conf. on Office Information Systems, Cambridge, Massachusetts, eds. Lochovsky F. H., Allen R. B., ACM Press, ACM SiGOiS Bulletin, Vol. 11, Issues 2, 3, Apr. 1990.

[Rich91] Richardson J., Schwartz P.: “Aspects: Extending objects to support multiple, independent roles”, ACM-SIGMOD Intl. Conf. on Management of Data, Denver, Colorado, ACM SIGMOD Record, Vol. 20, May 1991, pp. 298 - 307.

[Riet89] v. d. Riet R. P.: “MOKUM: An object-oriented active knowledge base system”, Data and Knowledge Engineering, Vol. 4, North-Holland, 1989, pp. 21 - 42.

[Rumb91] Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W.: “Object-Oriented Modeling and Design”, Prentice Hall, 1991.

[Scio89] Sciore E.: “Object Specialization”, ACM Trans. on Office Information Systems, 7(2), April 1989, pp. 103 - 122.

[Shil89] Shilling J., Sweeney P.: “Three steps to view: Extending the object-oriented paradigm”, OOPSLA’89, ACM SIGPLAN Notices, 24(10):353 - 361, October 1989.

[Stei89] Stein L., Zdonik S.: “Clovers: The dynamic behavior of types and instances”, Technical Report CS-89-42, Brown University, Nov. 1989.

[UML97] “UML Notation Guide”, Rational Software Cooperation, http://www.rational.com/uml, vers. 1.1, 1. Sept. 1997.

[Wier91] Wieringa R., d. Jonge W.: The identification of objects and roles - object identifiers revisited”, Technical Report IR-267, Faculty of Mathematics and Computer Science, Vrije Univeriteit, Amsterdam, December 1991.

[Wier94] Wieringa R., d. Jonge W., Spruit P.: “Roles and Dynamic Subclasses: A Modal Logic Approach”, Proc. of the 8th European Conf. on Object-Oriented Programming, ECOOP'94, Springer Verlag, Bologna, Italy, July 1994.

[Wong97] Wong R. K., Chau L. H., Frederick L. H.: “A Data Model and Semantics of Objects with Dynamic Roles”, Proc. of 13th Intl. Conf. on Data Engineering, IEEE CS Press, University of Birmingham, Birmingham, U.K., Apr. 1997, pp. 402 - 411.