

A Role-Based Metamodeling Approach to Specifying Design Patterns

Dae-Kyoo Kim, Robert France, Sudipto Ghosh, Eunjee Song
Computer Science Department
Colorado State University
Fort Collins, CO 80523, USA
{dkkim,france,ghosh,song}@cs.colostate.edu

Abstract

Design patterns describe solutions to recurring design problems in the development of software designs. To encourage the use of design patterns, we are investigating tool support for incorporating patterns into UML models. The development of such tools requires patterns to be specified at the metamodel level. Patterns may be specified using roles, where a role is played by model elements. However, the notion of role in the object-oriented community is strictly based on objects, and does not allow the use of the word “role” in any other place where the context is not object-based. In this paper, we propose a notion of role that can be used to specify design patterns at the metamodel level. We survey the characteristics of object-based roles and generalize them. Based on the generalized notion of a role we define a new notion of a model role which is played by a model element. We illustrate the use of model roles with a specification of a variant of the Observer design pattern.

Keywords: Design models, Design patterns, Objects, Roles, Unified Modeling Language.

1. Introduction

A design pattern [9, 12] describes a family of problem and solution pairs of the pattern, where applying the pattern to the problem results in the solution. To facilitate the incorporation of design patterns into design models, the development of tool support is necessary. The development of such tools requires precise representation of design patterns. We are investigating a metamodeling approach to specifying design patterns using roles.

The prevalent notion of role in the object-oriented community is defined at the model level, where a role is played by an object (henceforth referred to as object role). To allow the use of the word role in non-object based contexts, the notion of object role needs to be generalized.

Roles have been used in many other metamodeling approaches [4, 5, 10]. However, no work has been done on clearly defining the notion of roles at the metamodel level. In this paper, we define roles at the metamodel level to specify design patterns where a role is played by model elements (e.g., classes, associations). Such a role is referred to as model role. Using model roles allows tool developers to extend the mechanisms used to check conformance of UML models against the pattern specifications (metamodels).

We generalize the characteristics of object roles to be level independent. The generalized notion of role is referred to as general role. A model role is then defined as a specialization of the general role. Object roles and other roles at higher levels (e.g., metamodel level) can also be viewed as specializations of the general role as shown in Fig. 1.

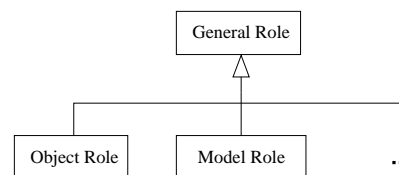


Figure 1. Roles

A model role has a base metaclass in the *Unified Modeling Language* (UML) metamodel [17]. For example, a model role whose base is *Class* metaclass is called a class role. The class role is played by classes (i.e., instances of *Class*) that satisfy the constraints specified in the role.

We give a historical overview of object roles, and survey the characteristics of object roles in Section 2. We generalize the characteristics in Section 3, and define model roles in Section 4. We specify a variant of the *Observer* design pattern using model roles in Section 5, and conclude in Section 6.

2. Object Roles

The word role was first introduced in the object-oriented community by Bachman and Daya [2]. Since then, there has been considerable work done on using roles for object-oriented data modeling [1, 3, 8, 11, 14, 15, 16, 18]. Sciore [14] and Gottlob *et al.* [8] address hierarchy of roles. Stein and Zdonik [16] and Pernici [11] use roles to describe object behavior. Wieringa and Jonge [18] argue that objects and roles should have their own identifiers. Steimann [15] uses roles to define an object-oriented modeling language that does not have a metamodel. Dahchour *et al.* [3] describe a general role model that consists of object classes and role classes where instances of object classes are instances of role classes.

The work on role modeling was used by the object-oriented design modeling community [7, 13, 17]. They emphasized the behavior of objects more than object-oriented data modelers had done with entities.

We identify five commonly accepted characteristics of object roles from the above work:

1. A role has structural and behavioral properties [3, 11, 16, 18]. Properties of a role may be inherited from other roles in a hierarchy [2, 8, 11, 14, 16, 18]. The hierarchy suggests that an object that plays a role can also play the superrole of the role. For example, a person who is a graduate teaching assistant also plays a graduate assistant.
2. A role defines a subset of objects of a type [2, 8, 18]. In other words, a role defines constraints on a type. Only those objects that satisfy the constraint can play the role.
3. An object may play multiple roles simultaneously [2, 11] or the same role several times with a different state during its lifetime [11, 8]. In the latter case the state of an object is role specific [11, 18]. For example, different phone numbers may apply to a person in different roles.
4. An object may acquire and abandon roles dynamically during its lifetime [8, 11, 14]. The sequence of acquiring and abandoning roles may be restrictive [11, 14]. For example, a person can become an advisor only after being a professor.
5. Access to roles is restricted by context. This means that a role is meaningful only in the context of a relationship [1, 11, 14]. For example, a student's grade is not accessible in the employment context if the person playing the student role also plays employee role [1, 8, 14].

3. General Role

We generalize the characteristics of object roles described in Section 2 to define general roles whose characteristics are independent of levels. We apply the following simple generalization rules to the characteristics of object roles:

- We exclude object level specific characteristics. Object roles are defined at the model level, and played by objects at the object level. Runtime specific characteristics described in items 3 and 4 in Section 2 are excluded.
- The word *object* in items 1, 2 and 5 is replaced by *instance*.

After applying the rules, general roles have the following characteristics:

1. A role has structural and behavioral properties. Properties of a role may be inherited from other roles in the hierarchy. Instances that play a role can also play the superrole of the role.
2. A role defines a subset of instances of a type.
3. Access to a role is restricted by context.

Specializations of the general role inherit the above characteristics, but may also have their own characteristics. An object role is a specialization of the general role in that instances are objects (e.g., instances of classes). Object roles have additional characteristics that are runtime specific. Theoretically, the general role can be used to define roles at any level (e.g., metamodel level).

4. Model Roles

In this section, we define model roles that are played by UML model elements (e.g., classes, interfaces, associations), not by objects. The UML infrastructure is defined as a four-layer metamodel architecture: Level M3 defines a language for specifying metamodels, level M2 defines the UML metamodel, level M1 consists of UML models specified by the M2 metamodel, and level M0 consists of object configurations specified by the models at level M1. Model roles are defined at level M2.

Fig. 2 shows relationships between model role and the UML infrastructure [17]. Every model role has its base metaclass in the UML metamodel. *MyRole* is a model role whose base is the *Class* metaclass. *MyRole* defines a subset of instances of the *Class* metaclass by adding constraints on the metaclass. *ClassA* that is an instance of *Class* is a member of the subset defined by *MyRole*, and the *ClassA* is said

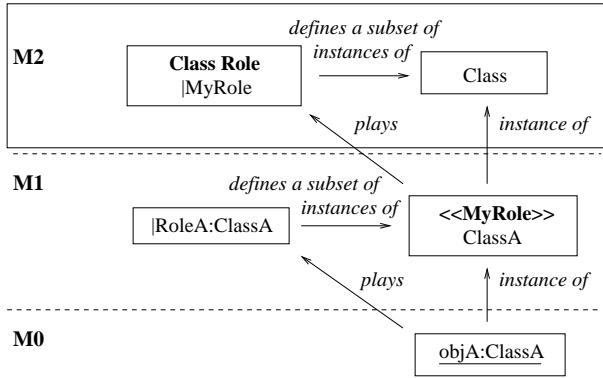


Figure 2. Relationship between Model Role and UML Infrastructure

to play the model role. We use the symbol “|” to indicate model roles.

Model roles inherit the characteristics of general roles, and have their own additional characteristics. The following are the characteristics of model roles:

1. Every model role has its base in the UML metamodel. For example, in Fig. 3 the *Subject* role has *Classifier* metaclass as its base indicated by a bold text label.

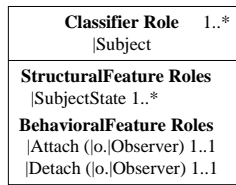
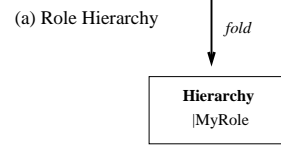
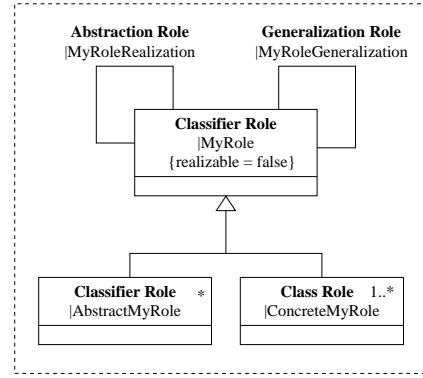


Figure 3. Example of a Classifier Role

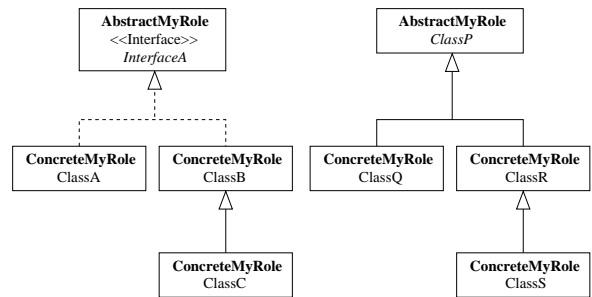
2. A model role defines a subset of instances of its base metaclass. For example, an association role specifies a subset of instances (i.e., associations) of *Association* metaclass. These instances are said to play the role.
3. A classifier role has structural and behavioral properties which are also roles whose bases are UML StructuralFeature and BehavioralFeature, respectively. In Fig. 3 *SubjectState* is a structural feature role, and *Attach* and *Detach* are behavioral feature roles. A classifier role is represented by a syntactic variant of the UML class symbol. For example, multiplicities are added to constrain the number of model elements that can play the role.
4. Properties of a model role may be inherited from a model role called *abstract role* which is not realizable.



(b) Folded Form of Role Hierarchy

Figure 4. Role Hierarchy

Fig. 4(a) shows a classifier role hierarchy. *MyRole* is an abstract role that comprises relationship roles (i.e., *MyRoleRealization*, *MyRoleGeneralization*) that are inherited by *AbstractMyRole* and *ConcreteMyRole*. The *MyRole* abstract role is simply an organizational entity and is not meant to be played. The hierarchy may be folded for high level view of abstraction as shown in Fig. 4(b).



(a) Conforming Model A

(b) Conforming Model B

Figure 5. Conforming Models of the Role Hierarchy in Fig. 4

Fig. 5(a) shows a conforming model of the hierarchy where

- *InterfaceA* plays *AbstractMyRole*, and

- *ClassA*, *ClassB*, and *ClassC* play *ConcreteMyRole*, and
- the realization relationship between *InterfaceA* and its realizations plays *MyRoleRealization* role in the hierarchy, and
- the generalization relationship between *ClassB* and *classC* plays *MyRoleGeneralization* role.

Fig. 5(b) is interpreted similarly except that the generalization relationship between *ClassP* and its subclasses play *MyRoleGeneralization* role.

- A model role may be played by several instances of its base metaclass. The number of model elements that can play the role may be constrained by multiplicities. For example, in Fig. 3 the multiplicity (1..*) in the top compartment indicates that the *Subject* role can be played by one or more classes. Multiplicity (0..*) allows roles not to be played by any model elements. For example, in the *Decorator* design pattern [9] the abstract *Decorator* class may be omitted when only one responsibility needs to be added.
- Model elements that play a model role may have application specific properties while satisfying the constraints defined in the role. Fig. 6(a) shows a class that plays the *Subject* role in Fig. 6(b). Dashed arrows point to the roles that the model elements play. For example, both *currPress* and *currTemp* attributes play the *SubjectState* role. The attribute *ready* and the operation *Activate* are additional application specific properties.

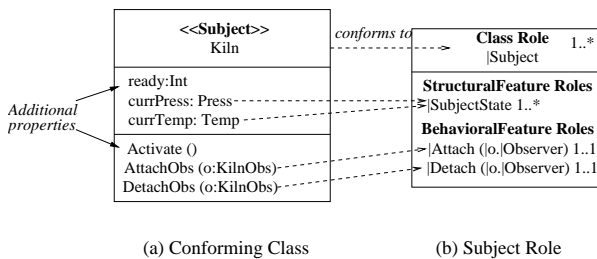
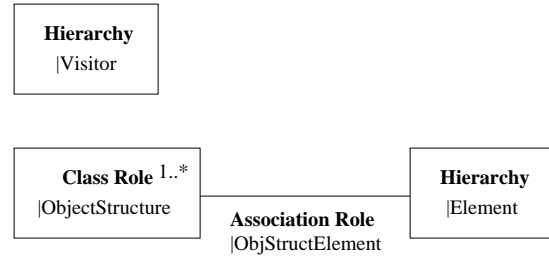
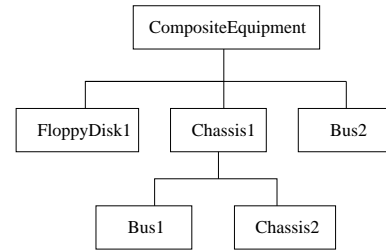


Figure 6. A Conforming class of the Subject Role in Fig. 3

- An instance of a metaclass may play several roles that have the metaclass as their bases. For example, consider the structure of an equipment in Fig. 7(b). When the *Visitor* design pattern [9] in Fig. 7(a) is used to calculate the total net price of the parts of the equipment, the class *Chassis* (not shown) plays both *ObjectStructure* and *Element* roles in Fig. 7(a).



(a) Visitor Pattern Specification



(b) An Object Structure of Equipment

Figure 7. Visitor Pattern Specification

- Properties of a classifier role can only be accessed in the context of the role. For example, The *Visitor* design pattern uses the *Composite* design pattern to describe the structure of elements [9]. In fact, the *Chassis* class in Fig. 7 plays four roles: *Element* and *ObjectStructure* roles in the *Visitor* pattern and *Component* and *Composite* roles in the *Composite* pattern in Fig. 8 (detail properties are not shown).

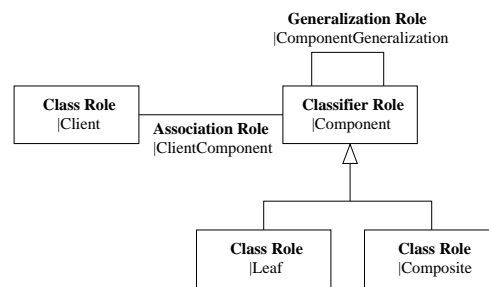


Figure 8. Composite Pattern Specification

Properties of the *Chassis* class that are specific to the context of the *Composite* pattern are not accessible in the context of the *Visitor* pattern.

The characteristics in items 2, 3, 4 and 8 are inherited from general roles; the rest are specific to model roles. The

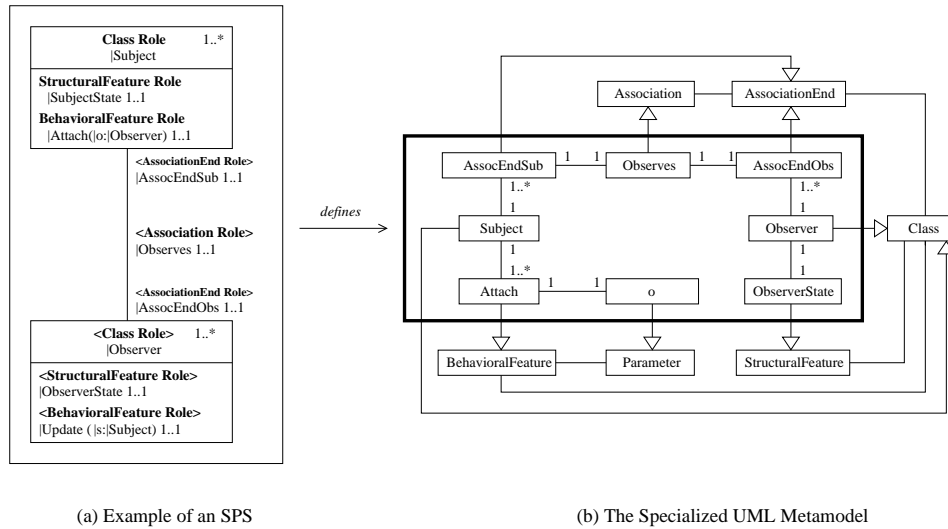


Figure 9. A Specification of Simplified Observer Pattern and its Specialized UML Metamodel

general role characteristic that states that “instances that play a role can also play the superrole of the role” is specialized in model roles; no model elements can play an abstract role (a superrole) in model role hierarchy (see Fig. 4(a)).

5. Static Pattern Specifications

We show a *Static Pattern Specification* (SPS) of a variant of the *Observer* design pattern given in the GoF book [9] using model roles. The SPS in Fig. 9(a) consists of two class roles, *Subject* and *Observer*, and an association role, *Observes*. The roles define subtypes of the base metaclasses in the UML metamodel, as shown in Fig. 9(b)¹. For example, the class role *Observer* defines a subtype of *Class* called *Observer* in Fig. 9(b). The *Subject* role defines the following constraints:

- *Role realization multiplicity - 1..**: A conforming static structural model must have at least one class that plays the *Subject* role.
- *StructuralFeature Role - SubjectState 1..1*: A class that conforms to the *Subject* role must have exactly one attribute or query that plays the *SubjectState* role.
- *BehavioralFeature Role - Attach(...)1..1*: A class that plays the *Subject* role must have exactly one operation that conforms to the *Attach* role.

A class diagram that conforms to the SPS must also have at least one class that plays the *Observer* role, that is, the

¹Fig. 9(b) only shows a partial view of the specialized UML metamodel.

class must have an attribute that plays the *ObserverState* role, and an operation that plays the *Update* role.

The association role *Observes* specifies associations between subject and observer classes. Each conforming association must have one association-end connected to a conforming *Subject* class and the other association-end connected to a conforming *Observer* class. The association-end connected to a subject class is specified by the *AssocEndSub* role, and the association-end connected to an observer class is specified by *AssocEndObs*. The multiplicity, *1..1*, on the *Observes* association role indicates that there can only be one conforming association between a conforming *Subject* class and a conforming *Observer* class. The multiplicity on the *AssocEndSub* role specifies that a conforming *Subject* class must be part of only one *Observes* association. Similarly, a conforming *Observer* class must be part of only one *Observes* association.

A simple class diagram that conforms to the *Observer* SPS is shown in Fig. 10. Dashed arrows point to the roles that model elements play. For example, the two association ends on *Kiln* play the *AssocEndSub* role on *Subject* role. The constraints for the multiplicities (1..1) on the associations ends are expressed separately using the Object Constraint Language (OCL) [17]. These constraints are not shown in this paper. The reader is referred to [6] for details.

6. Conclusions

We have described a metamodeling approach to specifying design patterns using roles. The approach is intended to be easy to use and practical for the development of tools that incorporate patterns into UML models. The

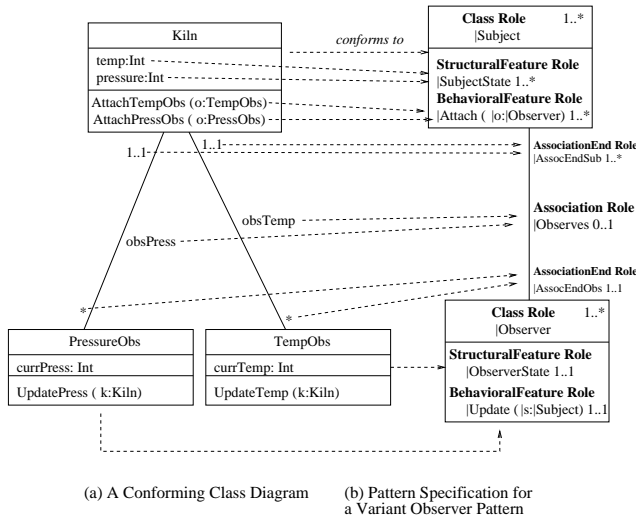


Figure 10. A Realization of the Observer Pattern in Fig. 9

notion of model roles has been also used to define other design perspectives of design patterns such as interactions (*Interaction Pattern Specifications*) and state-based behaviors (*State Machine Pattern Specifications*) [6]. To date we have developed specifications for the following GoF-based patterns [6]: Abstract Factory, Bridge, Decorator, Singleton, Observer, Composite, and Visitor. Our experiences indicate that specifying design patterns using model roles is no harder than designing UML models.

References

- [1] A. Albano, G. Ghelli, and R. Orsini. Galileo: A strongly-typed, interactive conceptual language. In *ACM Transactions on Database Systems*, 10(2), pp. 230-260, 1985.
- [2] C. W. Bachman and M. Daya. The Role Concept in Data Models. In *International Conference on Very Large Databases*, pp. 464-476, 1977.
- [3] M. Dahchour, A. Pirotte, and E. Zimanyi. A Generic Role Model for Dynamic Objects. In *Proceedings of the 14th Advanced Information Systems Engineering International Conference, CAiSE'02*, pp. 643-658, Toronto, Canada, May 27-31, 2002.
- [4] G. Florijn, M. Meijers, and P. van Winsen. Tool support for object-oriented patterns. In *Proceedings of the 11th European conference on Object Oriented programming, Springer LNCS 1241*, pp. 472-495, 1997.
- [5] M. Fontoura, W. Pree, and B. Rumpe. *The UML Profile for Framework Architectures*. Addison-Wesley, 2002.
- [6] R. France, D. Kim, E. Song, and S. Ghosh. Role-Based Modeling Language (RBML) Specification V1.0. Technical Report 02-106, Computer Science Department, Colorado State University, Fort Collins, CO, June, 2002.
- [7] U. Frank. Delegation: An Important Concept for the Appropriate Design of Object Models. In *Journal of Object Oriented Programming*, 44, pp. 13-17, June, 2000.
- [8] B. Rock G. Gottlob, S. Michael. Extending Object - Oriented Systems with Roles. In *ACM Transactions on Information Systems*, 14(3), pp. 268-296, July, 1996.
- [9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [10] A.L. Guennec, G. Sunye, and J. Jezequel. Precise modeling of design patterns. In *Proceedings of UML'00*, pp. 482-496, 2000.
- [11] B. Pernici. Objects with roles. In *Proceedings of the conference on Office information systems*, pp. 25-27, Cambridge, Massachusetts, April, 1991.
- [12] W. Pree. *Design Patterns for Object-Oriented Software Development*. Addison Wesley, 1995.
- [13] T. Reenskaug, P. Wold, and O. A. Lehne. *Working with Objects: The OORAM Software Engineering Method*. Manning/Prentice Hall, 1996.
- [14] Edward Sciore. Object Specilazation. In *ACM Transactions on Information Systems*, Vol. 7, No. 2, pp. 103-122, April, 1989.
- [15] F. Steimann. On the representation of roles in object-oriented and conceptual modelling. In *Data and Knowledge Engineering*, 35 (1), pp. 83-106, 2000.
- [16] L. A. Stein and S. B. Zdonik. Clovers: The Dynamic Behavior of Types and Instances. Technical Report CS-89-42, Department of Computer Science, Brown University, Providence, RI, November 1, 1989.
- [17] The Object Management Group (OMG). Unified Modeling Language. Version 1.4, OMG, <http://www.omg.org>, September 2001.
- [18] R. Wieringa and W. D. Jonge. The identification of objects and roles: Object identifier revisited. Technical Report IR-267, Vrije University, Amsterdam, 1991.