

Using Roles to Characterize Model Families

Robert B. France, Dae-Kyoo Kim, Eunjee Song, Sudipto Ghosh
Colorado State University, Fort Collins, CO 80523, USA

Abstract

The development of reusable artifacts often requires one to characterize families of problems and their solutions. This paper presents a metamodeling approach to expressing structural and behavioral properties of a family of solutions. The properties are expressed in terms of roles that can be realized by model elements. A model realizes a role model if the model elements associated with the roles possess the properties defined by their roles. A role model is essentially a characterization of a family of models.

Keywords: Design patterns, reuse, object-oriented models, role models, UML

1 Introduction

A pattern (domain) specification is defined here as a characterization of behavioral and structural properties of a family of solutions in a domain. A design model is said to *realize* a pattern specification if it satisfies the properties defined by the pattern specification.

Pattern specifications can be viewed as metamodels in the sense that they describe essential aspects of a family of models (the pattern realizations). A realization can have consistent properties not specified in a pattern specification (i.e., additional properties that do not contradict pattern properties), and the properties stated in a pattern specification can be realized in different ways across different model realizations.

The following concerns guide our work on developing a precise notation for expressing pattern specifications:

- Design models in the Unified Modeling Language (UML) [5] are becoming the norm in object-oriented development environments. The pattern specification notation must be capable of describing families of UML design models.
- The pattern specification notation should be based on the UML infrastructure to facilitate integration into UML-based tool environments. Concepts used in the pattern specification notation should relate to concepts in the UML infrastructure, that is, concepts defined at the object level (M0), the model level (M1), the UML metamodel level (M2), and the meta-metamodel level (M3). New concepts should only be introduced when existing UML infrastructure concepts prove inadequate.
- The pattern specification notation should allow a pattern developer to specify only those properties captured by the pattern in terms used to communicate the pattern concepts (e.g., a reader of the pattern specification for the Abstract Factory pattern should be able to clearly identify Abstract Factory, Concrete Factory, Abstract and Concrete Product concepts easily).

- The pattern specification approach should support a well-defined notion of pattern conformance. A modeler should be able to determine precisely that their models conform to the pattern specification. Furthermore, it should be possible to automate non-trivial aspects of the conformance checking procedure.

In this paper, we present an approach to precisely expressing structural and some behavioral properties contained in pattern specifications that begins to address the above concerns. The metamodeling approach presented in this paper allows one to create metamodels, called *Static Role Models* (SRMs), that characterize UML static structural models, that is, models that depict classifiers (e.g., UML class, interface, type constructs) and their relationships with each other (e.g., using UML association, dependency, generalization constructs) [5]. The approach is based on the notion of a *role*, where a role is a specification of properties that are realized in some form in realizations. The treatment of behavioral properties in this paper is limited to the use of constraints for expressing semantic properties are realized by operations in realizations.

In section 2 we give an overview of other works on precisely defining patterns and relate them to our approach. In section 3 we describe how SRMs can be used to document structural and some behavioral properties in pattern specifications and illustrate the approach using the Abstract Factory design pattern [2]. We conclude in section 4 with an overview of our plans to further evolve this work.

2 Related Works

Lauder and Kent [4] propose an approach to presenting patterns precisely and visually using graphical constraint diagrams. In their work, patterns are described in terms of three layers of models: *role-model*, *type-model* and *class-model*. A role-model describes the essential aspects of a pattern in terms of highly abstract state and behavior elements. A type-model is a refinement of a role-model in that it refines the role-model state and behavior elements in terms of types that abstractly specify domain realizations of the role-model. A class-model is a deployment of a type-model in terms of concrete classes. In their work, pattern realization is viewed as a refinement process in which a high-level pattern description is refined to a model realization. Establishing that a model conforms to a pattern (as expressed by a role-model) involves establishing refinement relationships across the model levels. Providing non-trivial automated support for such conformance checking requires support for proof generation/checking. Furthermore, the authors use a graphical form of constraints that is appealing but is not currently integrated with the UML and it is not clear how tools can support the notation.

Guenec et al. [3] use a metamodeling approach in which pattern properties are expressed in terms of *meta-collaborations* that consist of collaboration roles that are played by UML modeling constructs (i.e., instances of classes in the UML metamodel at level M2 of the UML 4-layer language infrastructure). They accurately point out deficiencies in the UML notion of role models and provide an alternative representation in terms of meta-collaborations that utilize a family of recurring properties initially proposed by Eden in [1]. Unfortunately, their paper does not describe how properties (other than hierarchical structures of classifiers) are specified in their approach, nor is there a clear notion of what it means for a model to satisfy a role model.

In our work role models are (meta) specifications that models realize. Like the Lauder and Kent work, type models can realize our role models and they can be refined to produce more concrete realizations. It should be noted that one can also create realizations of our role models that are

more concrete than type models. Unlike the Lauder and Kent work, the role model constraints are expressed in terms of the UML metamodel.

Like the Guennec et al. [3] work our approach is based on the UML metamodel. Unlike their work we discuss the form and nature of constraints that can be associated with these models. Our role models are also more concise and provide a more direct representation of pattern properties and structure that can be transformed into a more complex UML metamodel form.

3 Using Role Models to Express Pattern Specifications

In our approach, a pattern specification is expressed in terms of *Role Models*. A Role Model is a structure of roles, where a role is a specification of properties. A model construct that satisfies the properties specified in a role can *play the role*, that is, it is a realization of the role. Obtaining a realization from a Role Model involves realizing roles in the Role Model.

There are two types of properties that can be specified in a Role Model:

- *Metamodel-level constraints* restrict the form of constructs that can realize the role. These properties are expressed as constraints over the UML metamodel elements [5] and thus limit the syntactic form of constructs that realize the role. For example, in the Abstract Factory pattern the AbstractFactory role can have a metamodel-level constraint that restricts the model constructs that can play this role to abstract class, type or interface elements that consist of operations with particular signature forms.
- *Model-level constraint templates* are parameterized constraints whose instantiations express semantic properties that must be expressed in models that realize the pattern. An instantiated constraint template is referred to as a model-level constraint. For example, the Observer pattern specification can include a model-level constraint for the Subject role that states that an Attach behavioral feature results in a new reference (link or otherwise) to an Observer instance. The parameters of model-level constraint templates are role names. Instantiation of a role's model-level constraint template by realizations of the role parameters results in a model-level constraint. Formally, one has to show that the corresponding constraints expressed in a realization imply the pattern specification's model-level constraint. In our approach, the proof of implication is called a *proof obligation*, and must be discharged before one can state that a model is a realization of a pattern with model-level constraint templates.

This paper describes a type of Role Model called a *Static Role Model* (SRM). An SRM is a Role Model whose realizations are UML static structural models. An SRM depicts classifier and relationship roles, and static relationships among the roles.

A simple example of a SRM and a realization of the SRM are given in Fig. 1.

The SRM consists of two *classifier roles* (*Monitor* and *Subject*) and a *relationship role* (*Monitors*). Classifier roles contain properties that determine the form of classifiers that can realize the roles. Similarly, relationship roles define properties that determine the form of UML relationships (e.g., associations, dependencies) that can realize the roles. The classifier roles in Fig. 1 stipulate that only class constructs (instances of the *Class* metaclass) can play these roles, and the relationship role stipulates that only association constructs (instances of the *Association* metaclass) can play this role. The multiplicities on the relationship role are interpreted as follows: A class playing the *Monitor* role must have two associations playing the *Rel1* role that are connected to one or two classes realizing the *Subject* role; similarly, a class that realizes the *Subject* role can have zero or

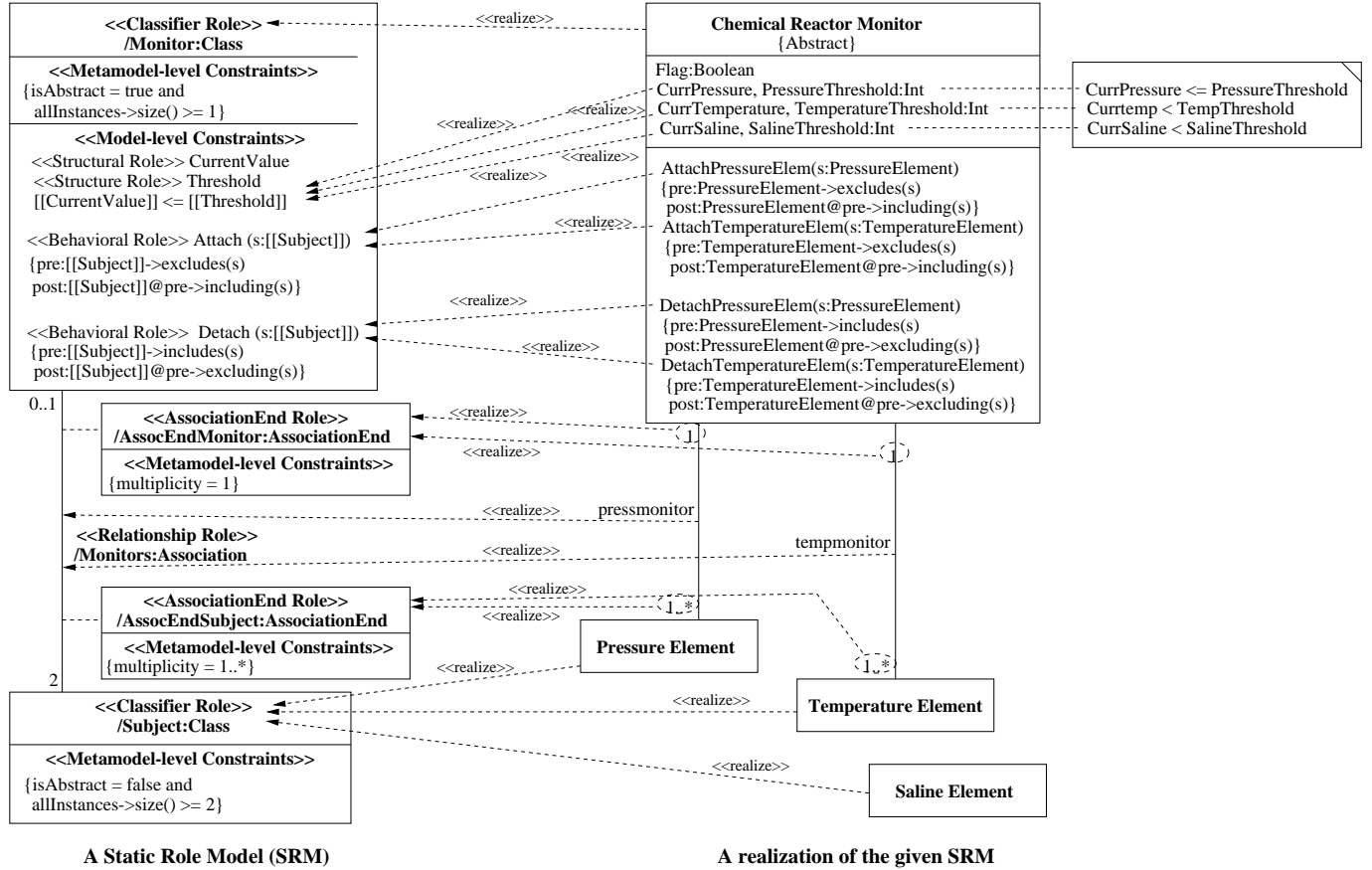


Figure 1: SRM for a Simple Pattern

one association that connects to a class realizing the *Monitor* role. Classifier and relationship roles are described in more detail below.

Classifier Role. A classifier role determines the minimal properties that classifier realizations must possess. In Fig. 1, *Monitor* and *Subject* are classifier roles that restrict their realizations to instances of the UML metamodel element *Class*. The metamodel-level constraints expressed in a classifier role constrain the form of the classifiers that can play the role. We use the OCL (Object Constraint Language) [5] to express these constraints. The metamodel-level constraint of the *Monitor* role states that a class playing this role must be abstract and there must be at least one class that plays this role in a realizing UML static structural model. In this constraint the implicit *self* term (e.g., as in *self.isAbstract*) refers to the *Class* class in the UML metamodel.

Model-level constraint templates are used to determine the semantic constraints that must be reflected in realizations of the classifier role. Two types of model-level constraint templates can be present in a classifier role:

- *Behavioral constraint templates* are used to define *Behavioral Roles* in terms of parameterized pre- and post-conditions. In Fig. 1 the *Monitor* role consists of two behavioral roles, *Attach* and *Detach*. The parameterized parts are indicated by roles enclosed within double square brackets ([,]). The *Attach* behavioral role states that a class playing the *Monitor* role must

define behaviors that support adding new instances of the classes playing *Subject* roles to the *Monitor* class instances.

A model-level constraint is obtained by substituting realizations for the parameters. Using the realizations shown in Fig. 1, an example of a model-level constraint is:

```

<< BehavioralRole >> Attach(s : TemperatureElement){
pre : TemperatureElement → excludes(s)
post : TemperatureElement@pre → including(s)
}

```

A single class operation or a sequence of classifier operations can realize a single behavioral role.

- *Structural constraint templates* each express a family of model properties on the state of the classifier. An example of a Structural constraint template is given in Fig. 1 and repeated below.

```

context [[Chemical Reactor Monitor]] inv
<< StructuralRole >> CurrentValue
<< StructuralRole >> Threshold
[[CurrentValue]] ≤ [[Threshold]]

```

In the above, *CurrentValue* and *Threshold* are structural roles defined in the *Chemical Reactor Monitor* classifier role. A structural roles can be realized by an attribute or an operation that returns a value and does not change the state of an object. The constraint template states that a realization of the *Chemical Reactor Monitor* must have an invariant that implies that the element playing the *CurrentValue* role has a value that is always less than or equal to the element playing the *Threshold* role.

Relationship Role. A relationship role determines the properties of classifier relationships that realizations must possess. The metamodel-level constraints expressed in a relationship role constrain the form of relationships in the realizations. For example, one can constrain the multiplicities associated with association-ends. Metamodel-level constraints are presented in terms of constraints on UML metamodel elements related to the relationship metaclass, and are expressed in the OCL. The constraint templates are structural constraint templates whose instantiations result in model-level constraints.

3.1 Role Models and the UML MetaModel

The structural properties expressed in a SRM can be expressed as a specialization of the UML metamodel. The UML metamodel expressing the structural constraints for the Role Model in Fig. 1 is shown in Fig. 2.

In the figure, classes that can play the classifier roles are instances of specializations of the *Class* metaclass. Specifically, classes that can play the *Monitor* role are instances of the metaclass *RoleA*, and classes that can play the *Subject* role are instances of the *RoleB* metaclass. Similarly, the associations that can play the *Rel1* role are instances of a specialization of the *Association* metaclass called *RelAssociation*. Metamodel-level constraints are expressed as constraints in the metamodel (not shown in Fig. 2).

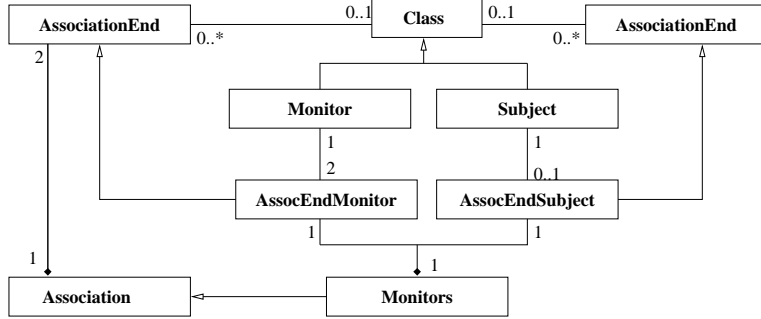


Figure 2: UML Metamodel view of Monitor Role Model

3.2 A Role Model for the Abstract Factory Pattern Specification

Fig. 3 shows the Abstract Factory pattern as presented in [2]. A problem with this representation is that it is a typical realization of the pattern and thus its use as a compliance point against which proposed pattern realizations are checked is very limited. For example, the AbstractFactory depicted in the diagram is an abstract class, but the example provided in [2] has an AbstractFactory realization that is a concrete class (the concrete operations provide default implementations). SRMs can provide better compliance points against which proposed realizations are checked.

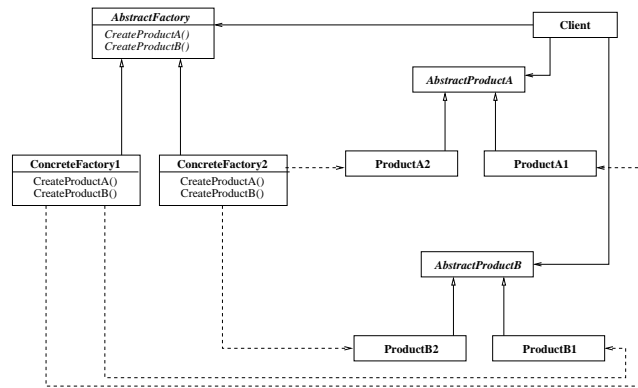


Figure 3: Abstract Factory pattern by GoF

A model is said to *realize a pattern* with respect to an SRM if the following holds:

- The model elements that are mapped to roles realize the roles. This means that (1) the model elements satisfy the metamodel-level constraints of the role, and (2) the constraints expressed in the realizing model (e.g., pre- and post-conditions for operations, and constraints on element attributes) conform to the model-level constraints derived from constraint templates (as discussed earlier).
- The model conforms to metamodel-level and model-level constraints expressed across roles.

It is possible to automate compliance checking against metamodel-level constraints (corresponds to syntactic checking of models). Compliance checking against instantiated model-level constraint

templates can be assisted by theorem-proving/checking tools, but it may not be possible to completely automate this activity. Modelers are obligated to discharge the proof obligations as part of the process of determining whether a model is a realization of a pattern.

The graphical part of an SRM expressing the pattern specification of the Abstract Factory pattern is shown in Fig. 4 (the metamodel-level constraints and model-level constraint templates are omitted). The SRM states that realizations of the AbstractFactory, AbstractProduct, ConcreteProduct, ConcreteFactory, and Client roles must be classifiers. Realizations of the AbstractFactory and ConcreteFactory roles are connected by relationships realizing the FactoryRel role. Similarly, realizations of the ProductRel relationship role connect AbstractProduct and ConcreteProduct realizations. Realizations of ConcreteFactory and ConcreteProduct roles are connected by realizations of the create dependency relationship role.

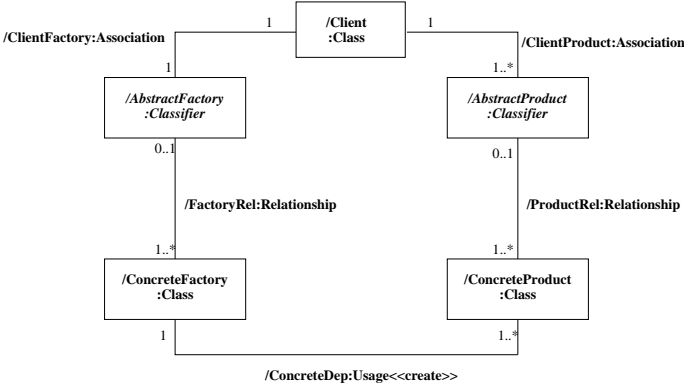


Figure 4: Abstract Factory SRM

The multiplicities on relationship roles depict the possible number of relationships that can exist between realizations of the classifier roles. It is important to note that the multiplicities on a relationship role are not the multiplicities that can appear on association ends in realizations. One can restrict the multiplicities on an association realization using a metamodel-level constraint defined for the association role.

Fig. 5 shows the metamodel-level and model-level constraints for the left part of the Abstract Factory pattern.

The structural aspects of the Abstract Factory SRM can be expressed as a specialization of the UML metamodel. An advantage that SRMs have over the UML metamodel representation is that SRMs are written in pattern-oriented terms rather than in generic UML metamodel terms. For example, Fig. 6 describes the left part of the Abstract Factory SRM that contains the FactoryRel relationship role between AbstractFactory and ConcreteFactory role.

3.3 Specializing Role Models

An SRM is a specialization of another parent SRM if it further restricts the properties specified in the parent SRM. Specialized SRMs allows one to present a pattern specification as a hierarchy of SRMs. As one goes down the hierarchy the set of models determined by the SRMs gets smaller. Fig. 7 shows three SRMs that are specializations of the AbstractFactory SRM shown in Fig. 4.

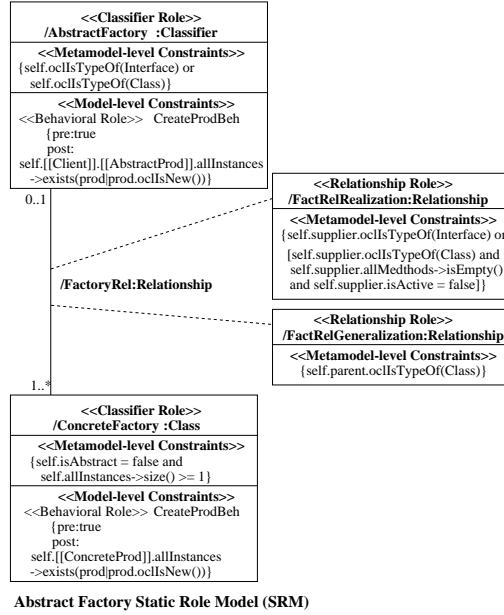


Figure 5: The left part of Abstract Factory SRM with constraints

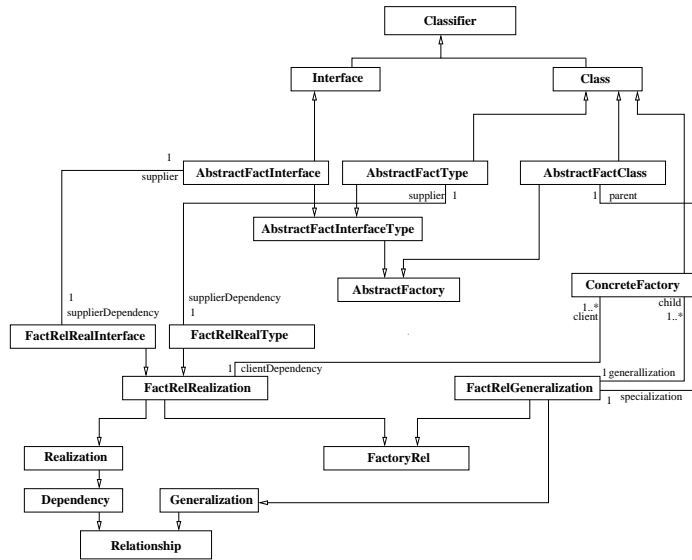


Figure 6: Meta-model for the left part of the Abstract Factory SRM

4 Conclusion and Future Works

In this paper we introduce the notion of Static Role Models (SRMs) as a means for expressing pattern specifications. SRMs are based on a well-defined notion of a role, where a role is a specification of properties that must be reflected in model-level constructs that realize the role. We are currently developing other types of role models that capture more complex behavioral constraints.

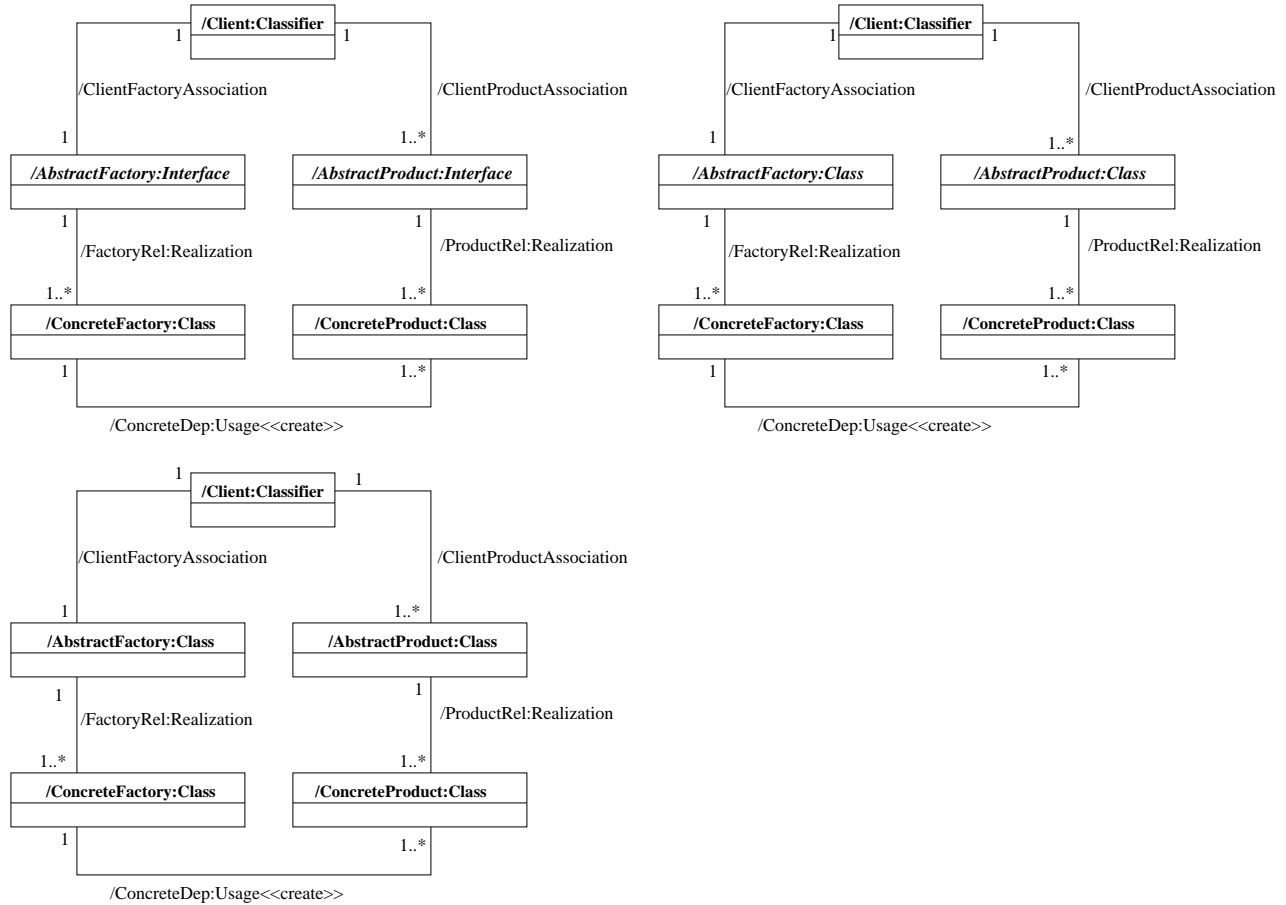


Figure 7: Abstract Factory Role Models represented by UML roles

Our current focus is on developing Meta Role Interaction Diagrams (MRIDs) to describe patterns of interactions at the metamodel level.

Our notion of roles can also be used as the basis for developing reusable models by incorporating role model elements into standard UML models (e.g., class diagrams, activity diagrams, collaboration diagrams). We refer to these models as hybrid role models (they consist of both roles and standard UML elements). For example, a reusable design class diagram can be created by using roles to indicate *model variation points*, that is, the parts of the design that must be tailored when reused. Tailoring the reusable model to a particular usage involves developing realizations of the roles and composing them as dictated by the reusable model’s structure. Roles can also be used to define design frameworks. The roles in a framework characterize the design elements that can be used within the framework. A particular design can be obtained from such a framework by plugging in realizations of the roles.

References

- [1] A. Eden. *Precise Specification of Design Patterns and Tool Support in Their Application*. PhD thesis, University of Tel Aviv, Israel, 1999.

- [2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.
- [3] A.L. Guennec, G. Sunye, and J. Jezequel. Precise modeling of design patterns. In *Proceedings of UML'00*, 2000.
- [4] A. Lauder and S.Kent. Precise visual specification of design patterns. In *Proceedings of ECOOP'98*, 1998.
- [5] The Object Management Group (OMG). Unified Modeling Language. Version 1.3, OMG, <http://www.omg.org>, June 1999.