JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 1, no. 4, September-October 2002

Modeling Roles
A Practical Series of Analysis Patterns

Francis G. Mossé, Object Discovery Corporation

While performing object oriented analysis, one often encounters problems related to *roles*. *Roles* are what any concept (or class) would play within the context of it's related concepts (or classes). For instance a "company" would be the "supplier" of some specific "product". "Supplier" is a *role*. All *role* problems can be easily solved by selecting one of the following 5 *role patterns*: Role Inheritance, Association Roles, Role Classes, Generalized Role Classes and Association Class Roles. Each *role pattern* contains its own blend of power, flexibility and complexity. Together, they offer a complete solution to all role problems.

Teaching an OOAD & UML course is a very rewarding experience. It makes course participants feel they can model problem domains they would never have tackled before. In assisting many corporations during their modeling effort, the challenge I've seen them encounter the most often is: the modeling of Roles.

Role problems are very frequent, and there exists at least half-a-dozen ways to solve them. We call them Role Analysis Patterns. They are "pre-cooked" solutions for typical Role analysis problems. Each pattern uses simple UML static modeling elements: classes, inheritance, associations and association classes.

After you read this article, you should feel that you can easily identify a Role problem and precisely determine which Role analysis pattern is best to model it.

1 THE INHERITANCE SOLUTION

One could easily think of Roles as *types*. For instance a *company* that *supplies products* would be understood as a *vendor*. On the other hand, if it *buys products*, then it's known as a *Customer*. That makes two different types of companies: v*endors* and *customers*. This concept could lead to the following model fragment (next page):

Cite this column as follows: Francis G. Mossé: *Modeling Roles – A Practical Series of Analysis Patterns*, in Journal of Object Technology, vol. 1, no. 4, September-October 2002, pages 27-37.
http://www.jot.fm/issues/issue_2002_09/column2

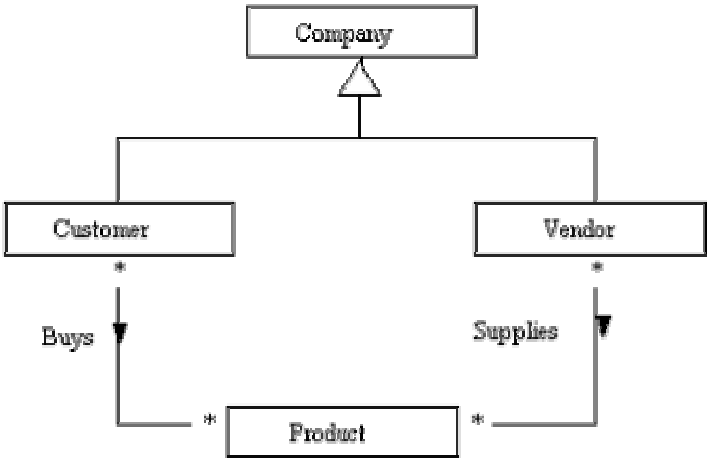


Figure 1: Inheritance Role Model Example

In *plain English* this model tells us that there exists two different types of companies: *customers* and *vendors*. *Customers* buy *products* whereas vendors supply them. Because inheritance is being used, this model also says that a company cannot be *both* a vendor and a customer; it's either one or the other—and once the choice is made, it's final.

Is this good or bad? It all depends on the problem domain you're trying to solve. Some business rules might dictate such a restriction; for example, one might not be allowed to get supplies from a customer. Other business rules might state the opposite: anyone may buy or supply at any time. We pick the model that best describes the role restrictions.

The model shown above is a concrete example of the inheritance solution. Now let's express the same solution for this example in an abstract way, using a *metamodel*. A *metamodel* provides a general answer, and models that describe a concrete example are *instances* of metamodels. Because metamodels supply a general answer, they are also called *patterns*.

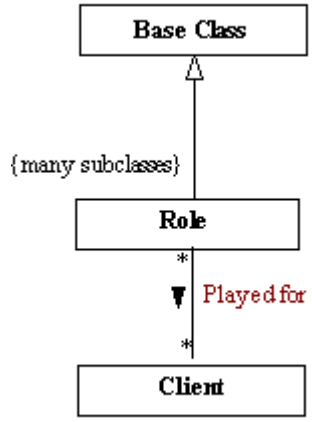


Figure 2: Inheritance Role Metamodel

Now, the inheritance metamodel disallows multiple roles to be played by the same base class. In other words, a company can be either a vendor or a customer, but not both. Similarly, a person can be either a dentist or a landlord, but not both! There are obviously cases where this limitation will be too restrictive to meet your problem domain requirements.

2 THE ASSOCIATION ROLE SOLUTION

Roles are names given to base classes (companies, persons, documents, …) *only* according to *what they do* in relationship to other classes. We could simply say*: a role is played by a class within the context of its relationship with other classes*. The UML Associations offer *Association Roles*, as depicted in the next figure. Notice that *Vendor* and *Customer* are simply association roles.

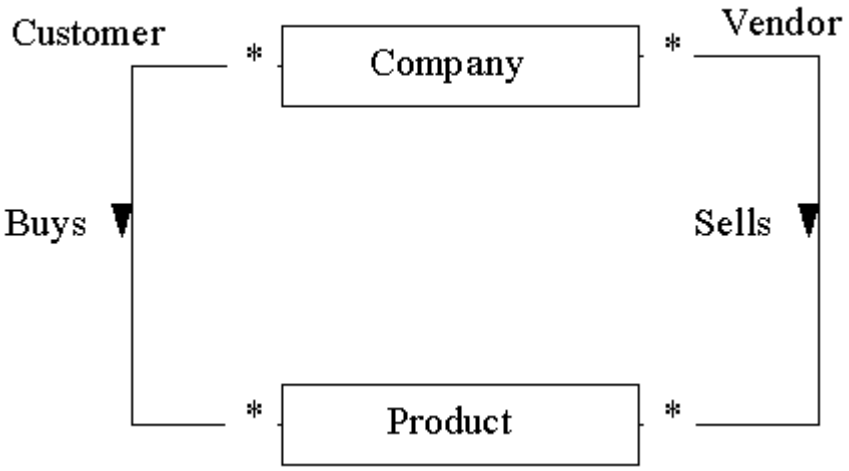


Figure 3: Association Role Model Example

According to the concept we started this new model with, a company's Role is simply based on what *it does*, not what *it is*. Figure 4 (below) is the metamodel for this second Role pattern.

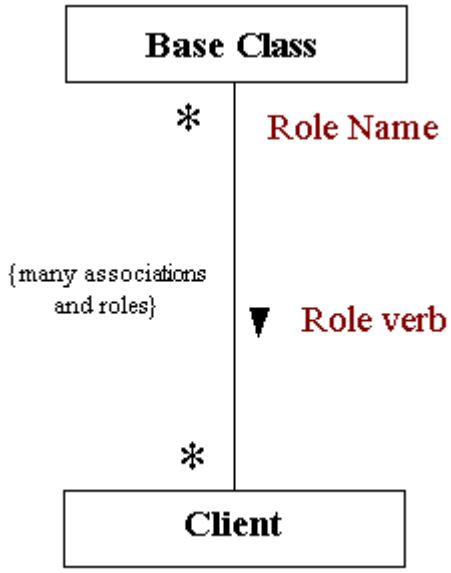


Figure 4: Association Role Metamodel

Figure 4 shows *Base Class* as an abstraction of classes like Company, and *Client* as an abstraction of Product. It allows for as many Base Class instances and Client instances as needed. The Role part of the metamodel is taken care of by the association and its own Role. The Association Role metamodel describes our *second role pattern*.

This Association Role pattern is clearly a very economical way of dealing with roles, but you'd be amazed to see how many problems can be solved with just that one feature. Since they are so efficient and useful, every modeler should be very familiar with Association Roles.

The following diagram (figure 5) shows another example using Association Roles. It also presents the complete notation for associations, including roles. This picture is an opportunity for the reader to become more familiar with the UML notation for associations and association roles.

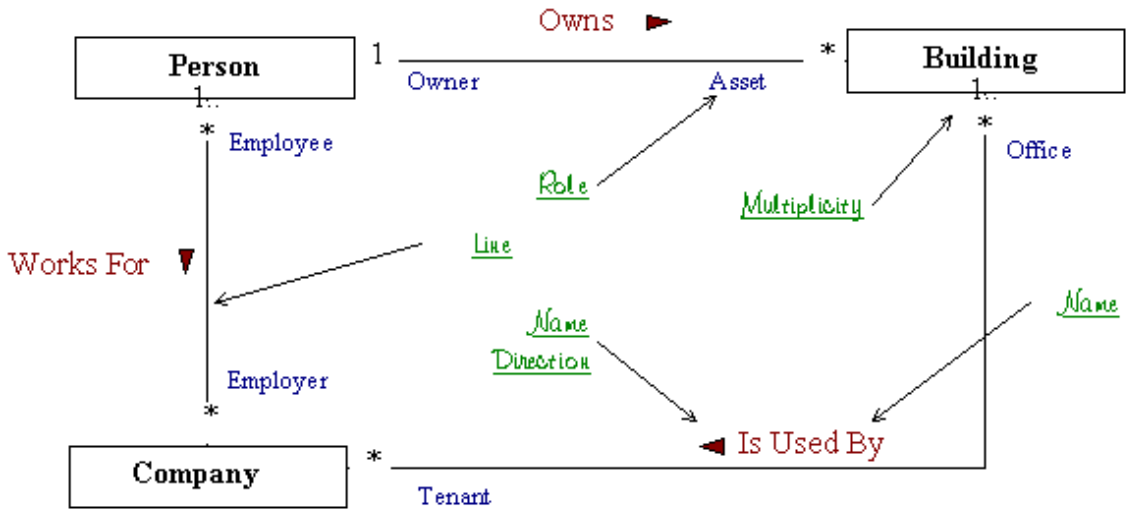


Figure 5: UML Associations general syntax, with Roles

An obvious drawback of Association Roles is the fact that they don't allow any attributes. Back to our original example, if the vendors or customers would require specific attributes, then we'd have to come up with a class Role solution, since classes can hold attributes. *Role Classes* are a way to accomplish that.

3 THE ROLE CLASS SOLUTION

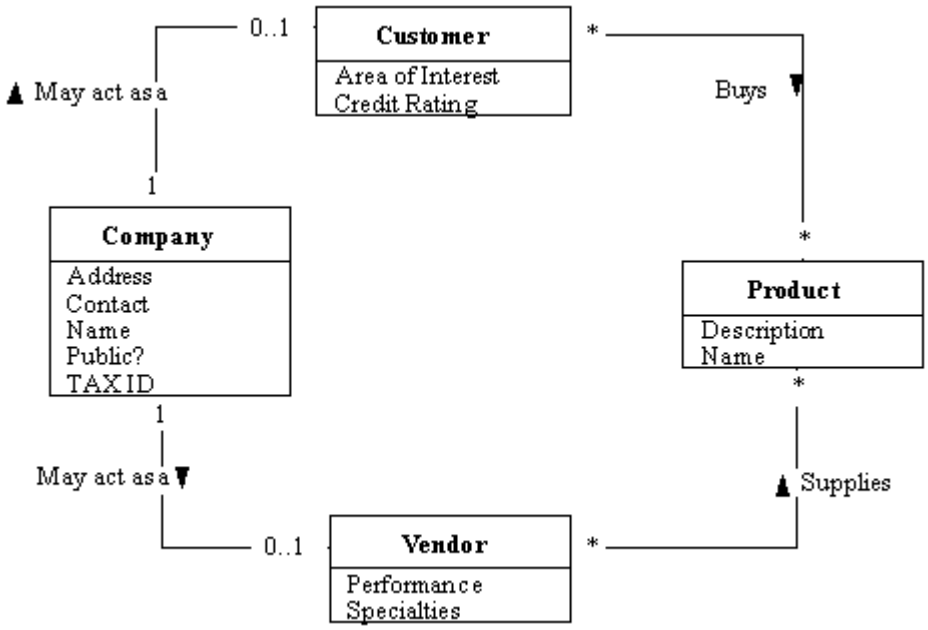


Figure 6: Role Class Solution Example

This model example allows you to link any company with any of the Roles it may possibly play.

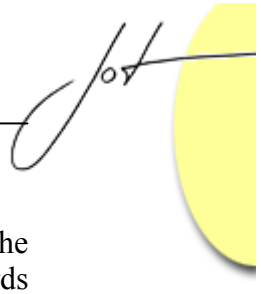
Here a role is modeled with a class—as it was in our inheritance solution in Figure 1, but this time no inheritance is being used.

This allows for flexibility of zero or multiple Roles while providing the power of Role-specific attributes and even class operations, if needed.

Figure 7 shows the Role Class Metamodel, our third Role Pattern.

Figure 7: Role Class Metamodel

This metamodel shows that the role class is like a middle man between the *Client* and the *Base Class*. Chances are that *Client* will "look" at *Role* as if it were the *Base Class* itself.



In our previous example (figure 6), a *Product* object might need the address of the *Customer* object that it's linked with. Then the *Customer* object will need to turn towards the *Company* object and get it's address from there. Delegation calls will be quite frequent. The same methods may be found in different classes (like "getAddress" and "getName" in Figure 8).

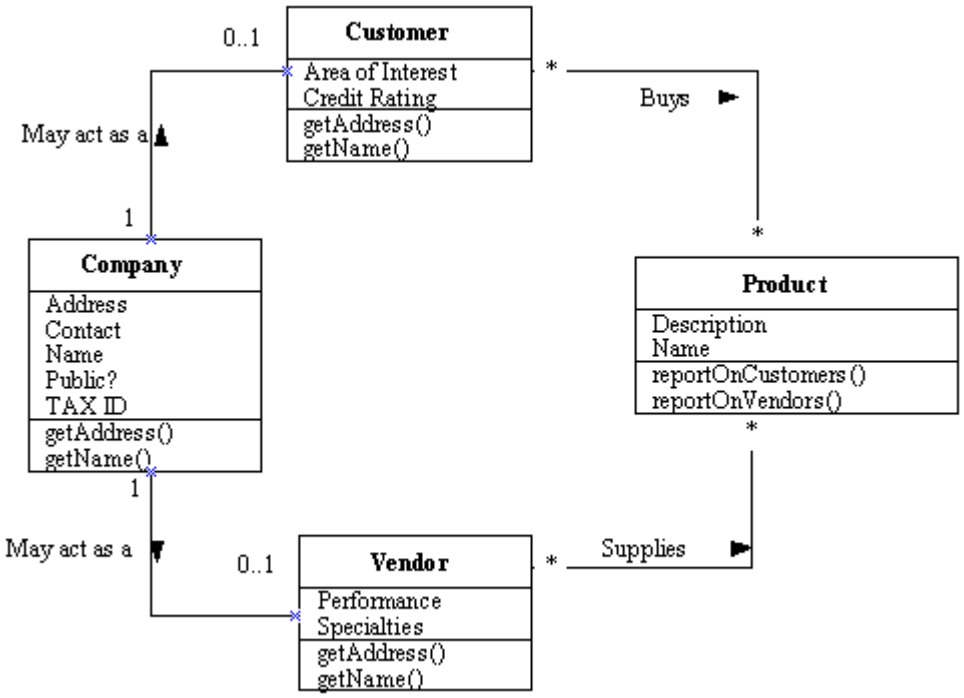


Figure 8: Role Class Model example with operations

4 THE ROLE CLASS GENERALIZATION SOLUTION

Figure 8 shows repetition of class operations; for instance, *getAddress() and getName()* operations are seen in both *Vendor* and *Customer* Role classes. Repetition calls for generalization. Figure 9 shows an example—along with an additional class role, *Broker*—that emphasizes the need for generalization.

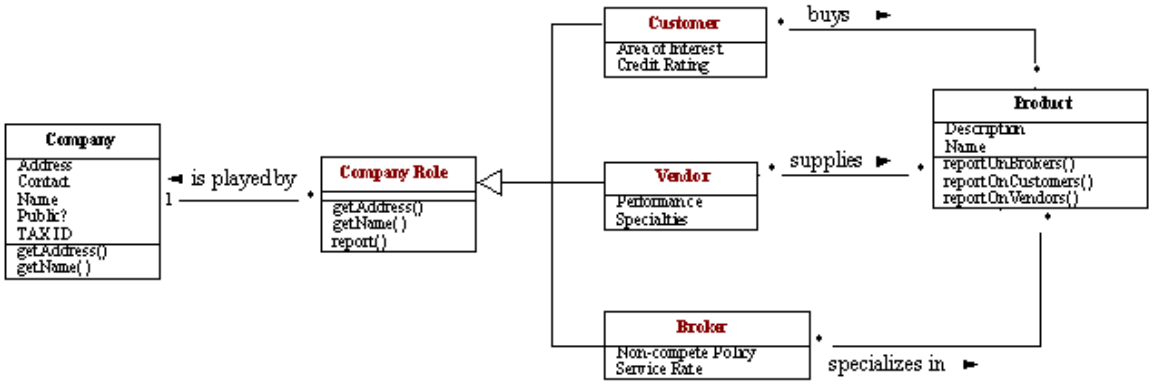


Figure 9: Role Class Model Example with Role Class Generalization

With the Role Class Generalization solution, all roles may be modeled and dynamically assigned to any *Company* at any time. Role-specific operations are supported but not repeated, thanks to the UML generalization feature. These Role-specific operations simply reside in one place: the *Company Role* generalization class. It is the one class that will take care of all delegations to Company. Only one implementation of its operations (methods) is needed, and it will serve all class Roles by virtue of inheritance. Figure 10 shows the Metamodel for this solution.

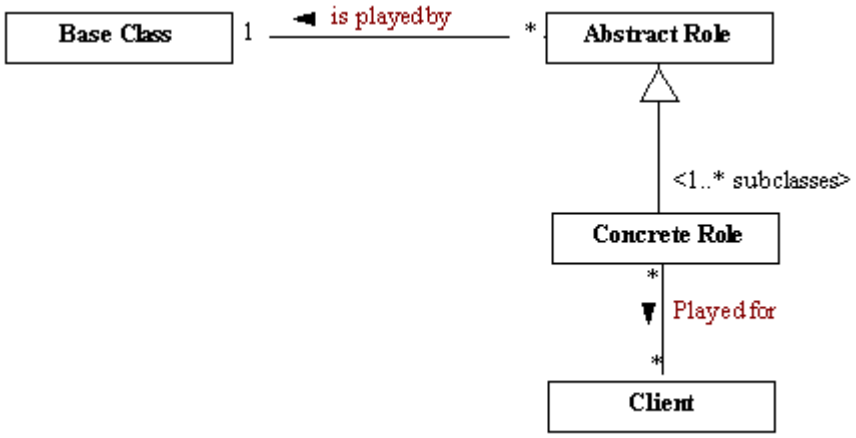
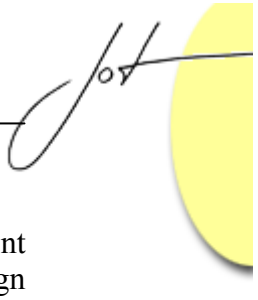


Figure 10: Role Class Generalization Metamodel

This metamodel describes the Role Class Generalization solution, our *fourth role pattern.*

5 THE ROLE ASSOCIATION CLASS SOLUTION

The examples we've seen so far assume that all roles are known at the time of the analysis. When the system is implemented, only *these* roles can be used. For instance, a



company can only assume the role of a vendor, a customer or a broker. These different roles can certainly be dynamically *assigned,* but not dynamically *created*. We can assign any role to Company, but it will have to be one of the roles already present in the system. In other words, it will have to be an existing class.

Classes aren't created dynamically—but *objects* are. The following example (Figure 11) shows how you can dynamically specify role relationships by using objects.

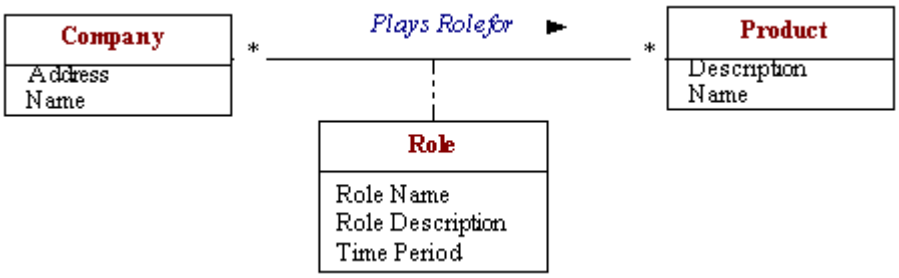


Figure 11: Association Class Role Example

In this case, roles are created on demand. To specify that a company supplies a specific product, then one just needs to create an instance of Role—a link object—and link the proper product to the right company. That role link object can be named whatever one desires it to be: vendor, customer, broker.

Note another interesting feature of this role association class: *Time Period*. Many business rules require relationship time periods to be specified.

The association class role solution is flexible and powerful. Notice that the *concept* of roles is modeled—not *roles* themselves. This model is more abstract than models that deal with explicit roles. In general, you will see that higher levels of abstraction bring not only greater simplicity, but also greater power and flexibility.

However, notice that more abstract models impose greater object manipulation. For instance, to solve the Vendor/Customer/Broker problem we looked in Figure 9, the application user would need to generate three link objects—one for each role.

An additional refinement needs to be brought to the association class role solution model. For instance, the model presented in Figure 11 would require generating multiple identical link objects for identical roles. For instance, 20 Vendor roles would require 20 Role link objects with identical name and description attribute values. Again, we encounter repetitions. Repetitions call for generalization.

This generalization cannot use inheritance since it needs be applied on *objects* as opposed to *classes*. Figure 12 shows a solution that also models the concept of generic and re-usable role types. Each role type can be dynamically created once as an object, and then linked to any number of role link objects that belong to that type.

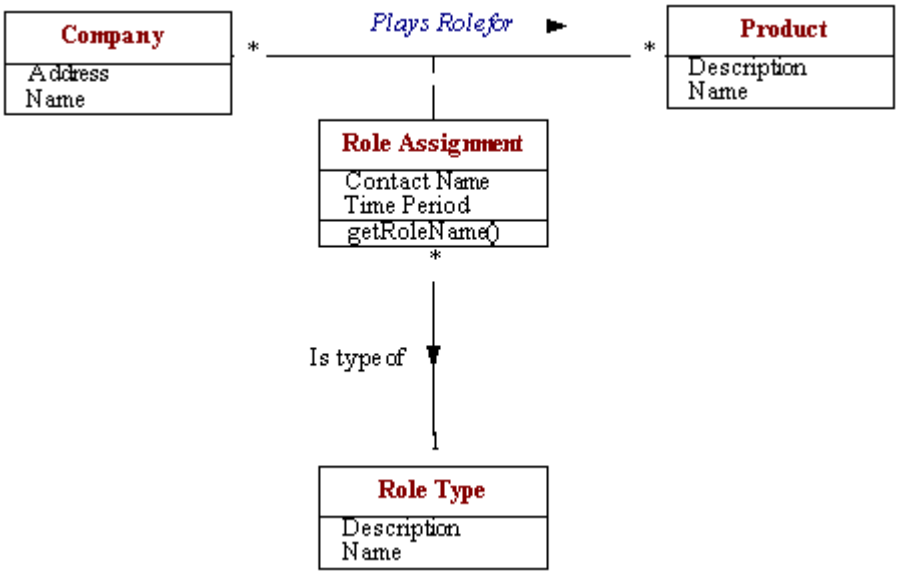


Figure 12: Association Class Role with Role Type Example

This solution allows for all Roles and Role Types to be dynamically created and used.

This model is abstract, powerful, flexible and object-intensive, as opposed to class-intensive.

In the earlier models you would need one class per role. In this new model, one class (Role Type) takes care of all cases, but each actual Role Type (e.g.: Customer, Vendor, …) still requires the creation of a Role Type object.

Figure 13 shows the metamodel for Association Class Role solution, our *fifth role pattern*.

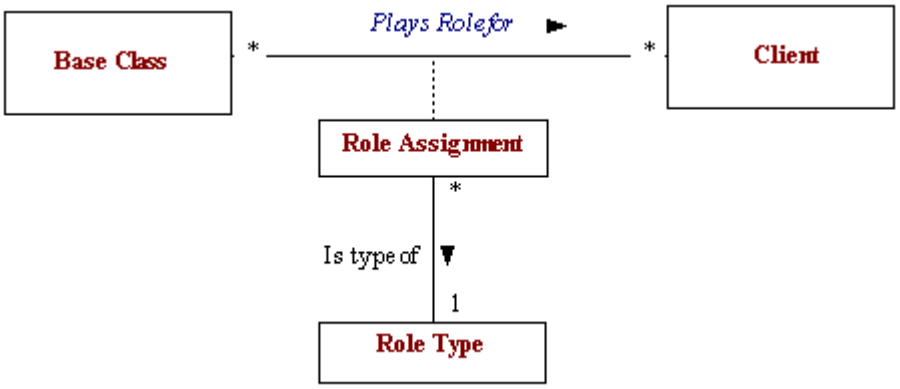


Figure 13: Association Class Role with Role Type Metamodel

6 CONCLUSION

We have seen a series of role requirements with their corresponding solutions, ranging from simple association roles up to role classes and role association classes. We've looked at the strengths and weaknesses of each solution. Role problems are very frequent. By knowing the solutions discussed here—also called patterns or metamodels—and their characteristics, one can quickly come up with the optimum model for any role situations.

About the author

Francis G. Mossé is an instructor and mentor in Object-Oriented Technologies at Object Discovery Corporation and can be reached about this article at: author@objectdiscovery.com.