

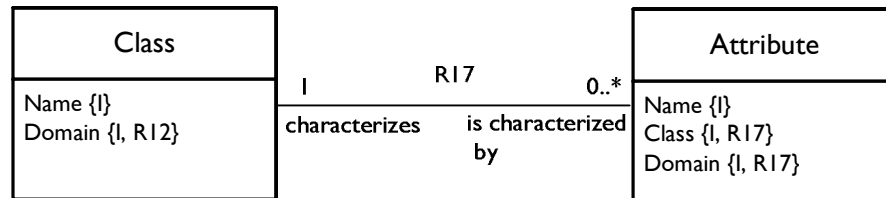
Modeling a simple thing

So there you are trying to model a bunch of rules involving a *thing* with very simple characteristics and behavior. This thing that you want to model seems like it should be modeled as a single class. But then life gets complicated. Important behavior, characteristics and constraints vary based on how and where this thing is used. And it becomes clear that a single class is just not going to cut it. Sounds like a classic generalization-specialization solution, but as we will see, subclassing doesn't always save the day. In fact, it rarely does. If patriotism is the last refuge of scoundrels, subclassing is the first refuge of the idiot.

Normally, I like to boil a teaching example down into cars and traffic lights or elevators and shafts or monkeys and bowling balls - you know, something we can all relate to. Unfortunately, I only found clear examples of this particular problem are in my metamodel projects. If you've worked with metamodels before this won't be any big deal and you can turn the page. For those of you new to metamodels, let me give you a quick orientation.

Brief explanation of metamodels

What's a metamodel? As you know, a class model formalizes classes, attributes and relationships in some subject matter. If the domain is elevator control, you might expect to see classes like shaft, cabin, door and so forth. A motor transport domain might be modeled with classes like motor, load, motion profile, stop, etc. Okay, now what if you are building software to manipulate UML models in some way? Maybe you are building a model editor or a model compiler that process UML class models. In this case your subject matter is the class modeling language itself. So you need to model elements of the UML class modeling language. You will end up with classes such as attribute, relationship and, yes, class. You would actually have a class that abstracts the notion of class. A model who's subject matter is a modeling language is referred to as a metamodel.



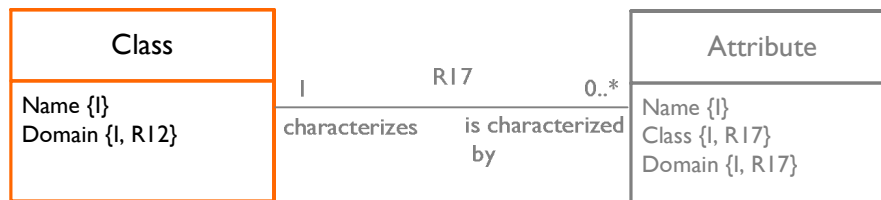
This is the kind of thing you would expect to see in a metamodel of the UML class modeling language. A metamodel is an ordinary model where the subject matter just happens to be the modeling language itself.



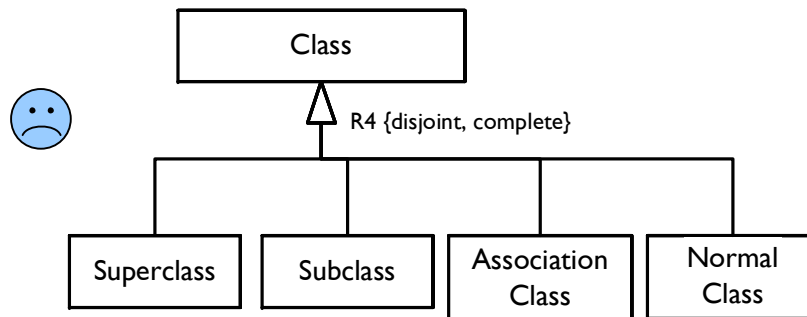
The Example

The thing, of which I spoke earlier, turned out to be the simple concept of "class". A class may or may not have a statechart. A class has zero or more attributes. A class participates in relationships. A class has a name and a description. It's easy to believe that we need a class called "class" to capture these concepts. So far so good.

The thing



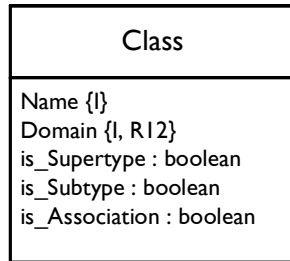
But wait a minute - what kind of class are we talking about? A superclass? A subclass? An association class? Or just a plain vanilla class that isn't anything special. Whatever that means. We could try specializing:



This model says that a given class must be one and only one of the designated specializations. But in a users UML model you might find a class that is both a superclass and an association class. Or both a superclass and a subclass. The same class can take on many combinations of roles.



Okay, how about just having one "class" class and slapping a bunch of boolean flags on it?



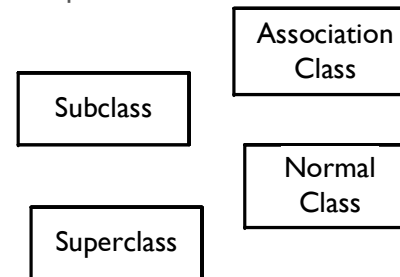
That's cute, but its not going to work. (Advanced modeler exercise: Pause and see if you can say why before reading on).



There are lots of constraints we need to model that will be impossible to capture without abstracting out the Association Class, Subclass, etc. as separate classes. An association class can only be an association class on one binary or reflexive association, for example. A Subclass may simultaneously participate as a Subclass in several generalizations. (Setting the isSubclass flag multiple times doesn't quite work for this concept!) A generalization must have exactly one superclass. The list of rules goes on...

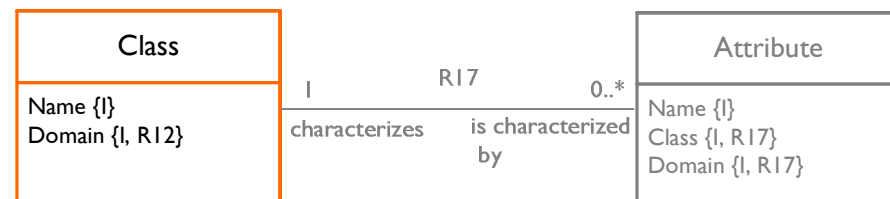
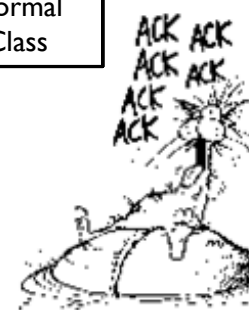
Okay, so we've established that we need the Class class. And we need the Association, Superclass, Subclass, Normal ... classes so that we can model the relevant constraints on each type. But somehow we must be able to navigate from a particular instance of an Association Class to its plain old Class representation. But generalization won't work. Acck! What to do?

1 We need to separate these guys out, so that we can draw constraining relationships on them.



3 But without using generalization, how do we navigate from one to the other?

2 And we need this stuff since it represents what all classes have in common irrespective of role.

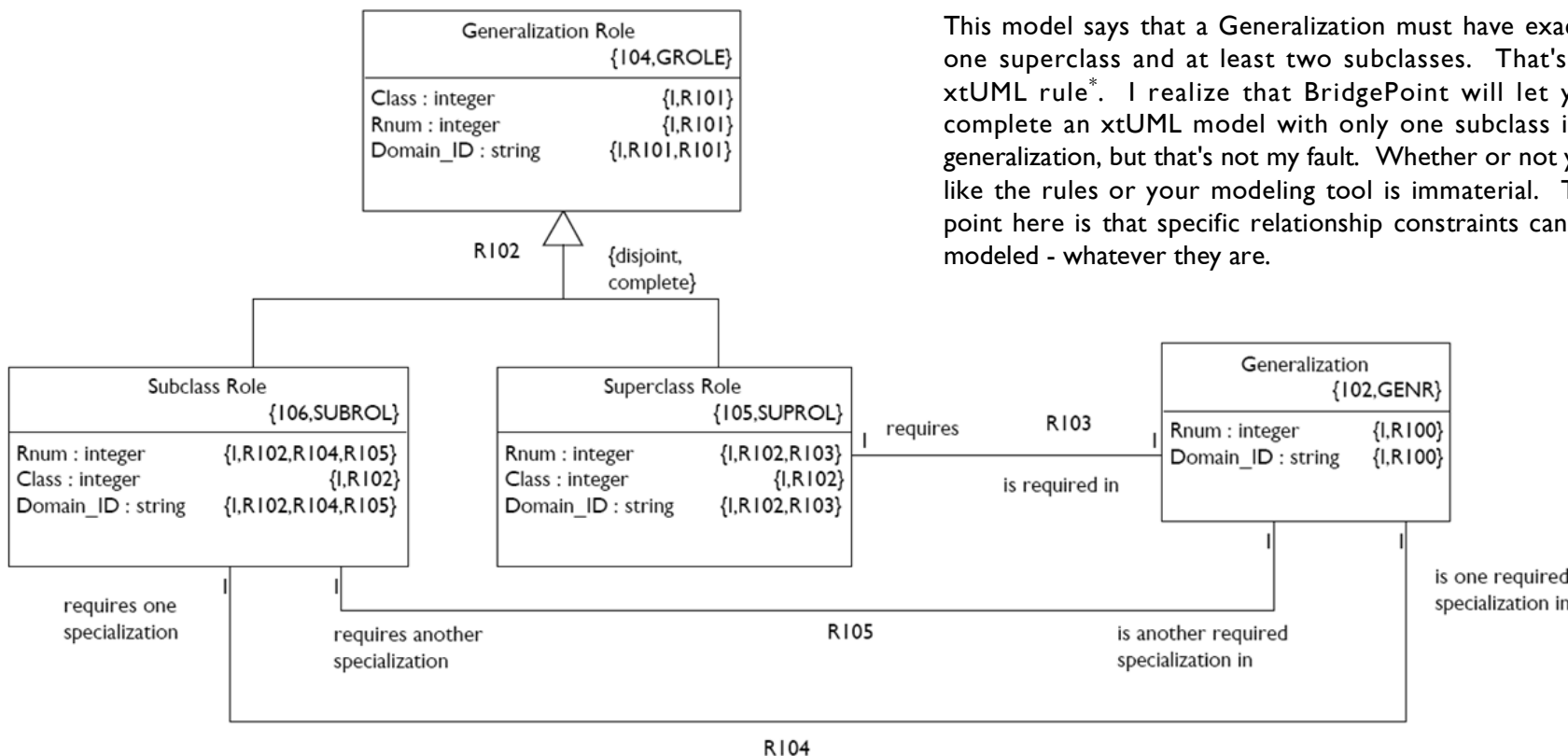


Model Puzzle 1 - Complex Roles

Leon Starr
mint.puzzle.1



A good technique to use when you encounter a weird modeling problem is to identify what model elements you want without worrying, for the moment, about how you're going to connect them to, or derive them from the rest of your model. Let's say, for example, that we have the Superclass and Subclass classes and forget, for the moment, how we will connect them back to the Class class. What would we do with these classes to capture the constraints of an xtUML generalization relationship?

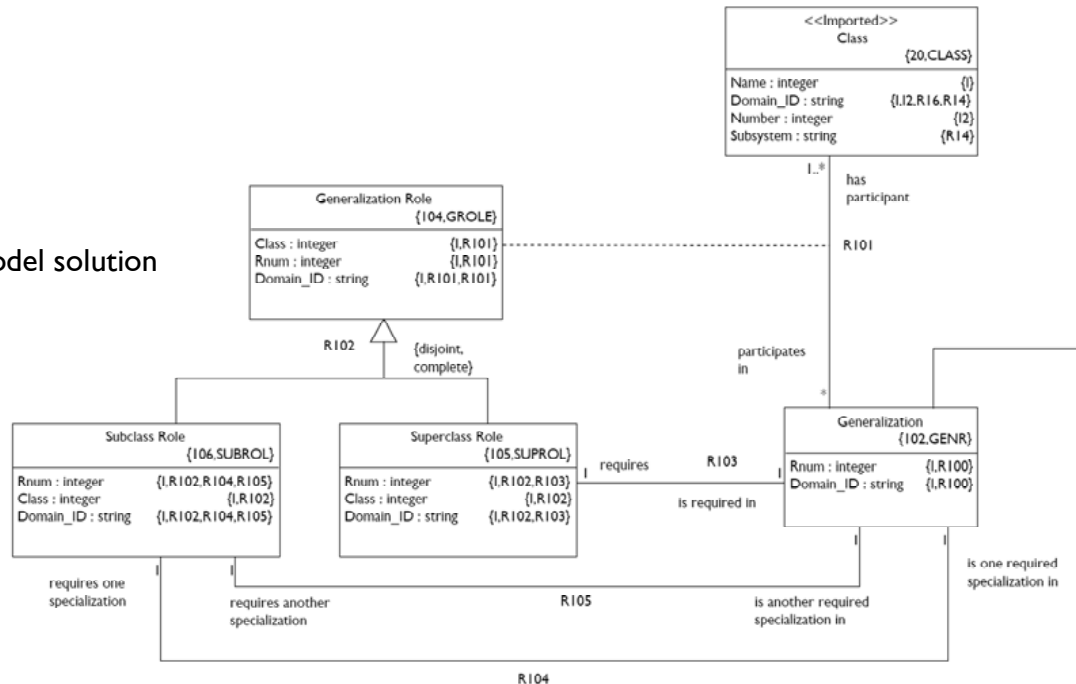


This model says that a Generalization must have exactly one superclass and at least two subclasses. That's an xtUML rule*. I realize that BridgePoint will let you complete an xtUML model with only one subclass in a generalization, but that's not my fault. Whether or not you like the rules or your modeling tool is immaterial. The point here is that specific relationship constraints can be modeled - whatever they are.

* See section 6.5 of Executable UML (Steve Mellor, Marc Balcer) or Chapter 6 of Executable UML: How to Build Class Models (Leon Starr) for the xtUML generalization constraints.

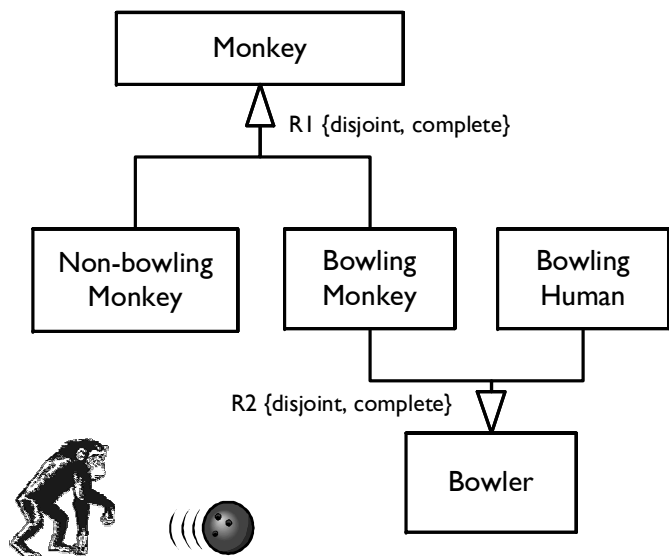


② ... into our metamodel solution



①

To see how the solution works, let's stuff this example user model fragment...



③ ...to get these instances.

Generalization Roles

- R1:Monkey
- R1:Non-bowling Monkey
- R1:Bowling Monkey
- R2: Bowler
- R2:Bowling Monkey
- R2:Bowling Human

Supersclass Roles

- R1:Monkey
- R2: Bowler

Subclass Roles

- R1:Non-bowling Monkey
- R1:Bowling Monkey
- R2:Bowling Monkey
- R2:Bowling Human

Classes

- Monkey
- Non-bowling Monkey
- Bowling Monkey
- Bowler
- Bowling Human

Generalizations

- R1
- R2

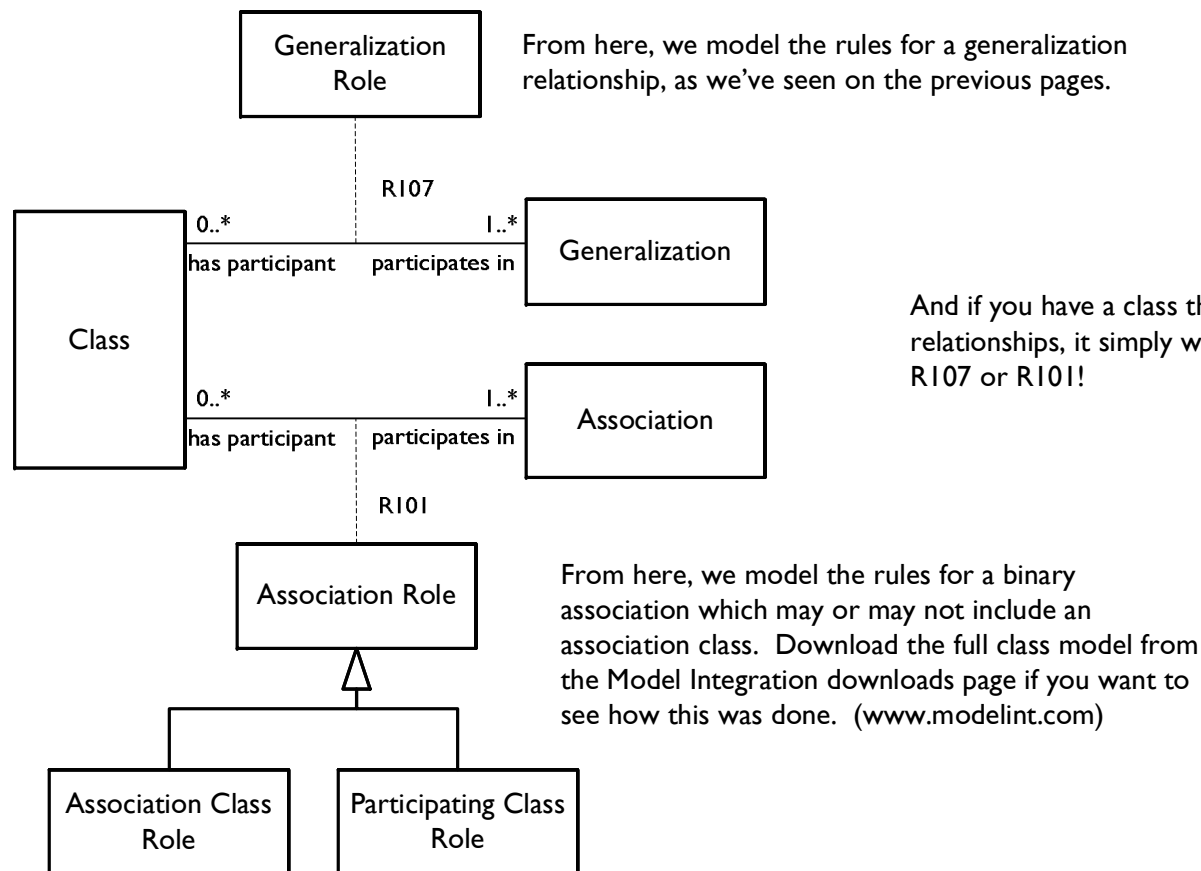
Model Puzzle 1 - Complex Roles

Leon Starr
mint.puzzle.1



Okay, that works. But what about association classes and classes that don't participate as either association classes or super/sub classes?

We just use the same technique to abstract a role to represent the participation of a class in a relationship.



From here, we model the rules for a generalization relationship, as we've seen on the previous pages.

And if you have a class that doesn't participate in any relationships, it simply won't have any links along R107 or R101!

From here, we model the rules for a binary association which may or may not include an association class. Download the full class model from the Model Integration downloads page if you want to see how this was done. (www.modelint.com)

Hope this helps! If you have any questions or comments, please post them in the "Modeling Problems" discussion group at www.modelint.com under the Resources page.

