Using Strategy Design Pattern with EOF

Work In Progress

Introduction

EOF and Entity Modeler allow you to model class inheritance using a number of strategies for mapping the inheritance hierarchy to the database. A complex EOF inheritance hierarchy can sometimes be challenging to implement, unwieldy to maintain, difficult to refactor, complex to add features, reveal buggy behaviour in EOF, and overall can take the Fun out of your development! Not only that, the Strategy Design Pattern helps us to build more maintainable code by implementing the following OO Principle:

OO Principle

Favor composition over inheritance

To understand how the Strategy Design pattern works, see chapter 1 of the Head First Design Patterns book. One will notice that in theroy the Strategy Design pattern is proposed as a good solution to dynamically change behaviour of objects, and this is certainly true, however the principle of object composition that is used in the common Strategy Design pattern can be applied to Entity modeling as a way to avoid the cumbersomeness and inflexibility of inheritance which are particularly apparent as an application's requirements evolve, change and grow over the lifetime of the application. Also the strategy design pattern is typically used for behavior change and not the availability of attributes, however this example will show how attributes can easily be managed, allowing different instances of the same entity to behave like it has the subset of attributes that apply to the instance type at hand, especially when it comes to displaying and editing in WOComponents.

I am sure there are many ways to skin a cat. Here we will look at one way to implement the Strategy Design pattern for a single Entity.

What Are We Going to Model?

We are going to model Parts in a Manufacturing/Production business.

We have the following types of parts which are all going to be represented by one entity named Part:
FinishedAssemblyPart
SubAssemblyPart
PurchasedPart
RawMaterialPart

This example will present a nice challenge since it has recursive to-many relationships back to itself.