Re: Confusion about splitting classes to allow sharing of resources

Source: http://coding.derkeiler.com/Archive/General/comp.object/2005−04/msg00449.html

* *From*: "H. S. Lahman" <h.lahman@xxxxxxxxxxx>
* *Date*: Fri, 22 Apr 2005 17:34:35 GMT

```
Responding to M.a.stijnman...
```

```
I agree with the flaw for a number of reasons. As
you note, lots of
different algorithms can be applied to the same
data. Sorting out and selecting algorithms is a
good job for patterns like Strategy. That
allows the selection of the algorithm (i.e.,
dynamically instantiating> the relationship) to be
separated from the routine computations.
Generally that selection will require an
understanding of the context> that is beyond what
the individual objects involved in a specific
collaboration need to know about (i.e., they have
there own problems to resolve).
```

```
That should be easy enough to do by having a hierarchy of concrete
Spline classes, such as LinearSpline and CubicSpline, based on the same
Spline interface. The higher-level class Curve should be able to select
which to use for a given implementation.
```

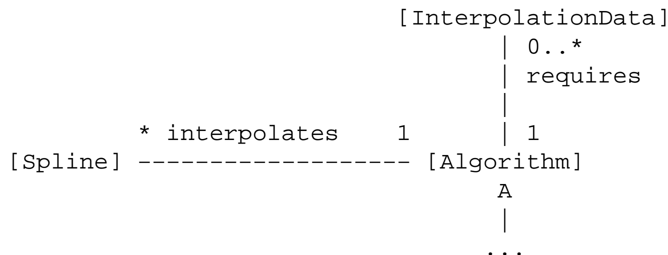
```
One thing you might consider is having a generic Spline and delegating
both the algorithm and the data (separately). Then the Spline would
handle the higher level "walking" of points and would delegate the actual
interpolation to the algorithm de jour one set of points at a time.

As I understand the problem description so far, the only differences
between spline flavors are (a) the interpolation algorithm, (b) the
interpolation data a particular algorithm needs, and (c) the data
associated with a point (maybe).

The interpolation algorithm you can delegate out via the GoF Strategy
```

pattern. The interpolation data vectors can be instantiated on an as-needed basis once the algorithm is selected. Since only the algorithm accesses that data, the algorithm will know how to navigate the vectors. If the data associated with a spline point (other than the interpolation data for a specific algorithm) can be different, then the Node can be subclassed.

> Separating the data also allows one to apply parametric polymorphism. One encodes more generic algorithms and allows detailed problem differences to be described in data. That, in turn, allows the problem structure to be define in external configuration data rather than being "hard-wired" in the code. Separating the data also allows one to
> capture the details of the problem in static structures. For example, the number of knots and function values can vary from one problem to the next while the algorithms remain the same (i.e., they "walk" the data collections in exactly the same way).
>
>> Whenever some of the data is changed, the spline object sets a flag so that when next time interpolation is performed, the 'behind-the-scenes' interpolation data gets regenerated. What you are telling me here, if I understand correctly, is I should think of spline as an algorithm, rather than as an object. Probably my data to be interpolated should then also be stored outside of the spline instance. Is that right?
>
> Maybe. B-) I'm not advocating functions as first class objects so one still has objects for the algorithms a la the Strategy pattern. What I am advocating is separating the concerns and encapsulating them in
> multiple objects. Then use the relationship structure instantiation to define the specific problem. So in that sense your conclusion is correct; delegate the data responsibilities elsewhere and isolate the
> algorithms that may be substituted for processing the data.

> Fact remains that the Spline may need to store state data for the interpolation. A cubic spline, for instance, needs to store a vector of the same length of the two input vectors - I usually refer to that as the interpolation info. If the contents of any of these change, the info becomes invalid and will need to be regenerated before any next interpolation can be performed. You are spealing about responsibilities. In the current situation, the spline itself is responsible of keeping an eye out for changes in the data. From the viewpoint of a user of the Spline class that seemed quite convenient to me. But decoupling the data and the interpolation calculation seems to imply that responsibility does not belong there. But this seems counterintuitive to me - my intuition tells me to hide as much of that sort of details away inside a class interface, so the users of the class don't have to worry about whether or not the spline is valid or not - the Spline object takes care of that itself. Where is the way out of this apparent contradiction?

There seem to be two distinct issues here. One is that special interpolation data is needed for certain algorithms or splines. (I am not clear where the dependency is, but that just determines where the relationship is rooted.) Let's say it is related to the selected algorithm. Then one might have:



The [InterpolationData] elements are instantiated when one knows what algorithm is used. The Algorithm knows it needs that data so it just "walks" that collection once it is instantiated (i.e., the algorithm can be confident the data structure it needs is there).

The second issue is synchronization for data changes. I am unclear what data is changed (the position Nodes on Spline?) and who is changing it (the point-to-point interpolation Algorithm?). When the data is changed, then the relevant [InterpolationData] elements need to be regenerated. [Algorithm] seems like the logical owner of the regeneration behavior since it is the only one that uses the data.

However, somebody has to know when the data is changed. Whoever that is can trigger the regeneration of the [InterpolationData]. (Alternatively, they can just set a semaphore for [Algorithm] so that it can regenerated

on an as-needed basis.)

> > I think the schema can be a little simplified in my case, since both the position (x,y) as the parameters p and q are modelled as a parametric function of a single parameter t. The exact shape of the function is determined by the values of x, y, p and q at the curve nodes, as well as the value of t defined at the curve nodes. This vector t[i] is what I refer to as the knot vector and is what is used to build the splines. So there is only one knot vector, even though each variable can have it's own type of interpolation.
>
> OK. I inferred you were looking for something more versatile where one could add other properties besides p and q in a fairly arbitrary manner. If not, you only need one data holder class for the curve points.

The number of properties will not vary at runtime, only by subclassing a Curve, so yes, I think one data holder should be sufficient.

I would be inclined to subclass the data only (once it was separated from Spline). That focuses on where the differences really are.

For example, there is an interface problem in accessing the properties if they have have different semantics, types, etc. Whoever accesses the data needs to get the right interface. One way to deal with that is something like the Visitor pattern. Another way is to provide a "reader" facility. In any case one needs to do something special to access different properties in different contexts. I think that will be easier to manage if the data is isolated from Spline.

> > As for the adding of nodes and knots, you're saying that should be handled by whatever class wants to

> instantiate a Curve or a Spline,
> right? Especially for the splines that
> makes much more sense if I see the spline
> class as an algorithm on outside data,
> not as an object
> holding data. So I should write a
> SplineFactory class, and a
> CurveFactory class, to provide that sort
> of facilities, correct? I
> think I'll read up on factory classes a
> bit then, figure out how best to
> implement them.

Yes, pretty much. I think the problem is
complicated enough to warrant separating the
concerns of constructing the overall structure into
a
dedicated object(s). This is especially true if you
use data holders and relationship collection
classes for your [i] data. That object can
understand and encapsulate the rules for properly
instantiating the
relationships.

I am not sure you need a bunch of factory objects
(e.g., one for each class), though. It seems like
the structure is pretty fixed for a given curve. In
that case it is pretty much just a succession of
constructor calls. IOW, it is complicated enough to
encapsulate in a dedicated
object but not so complicated that it needs to be
delegated among
multiple objects. The main decision seems to be
determining the right algorithms to use given the
Curve type. If that is not too complicated, it
might be easier to jam it all in a single factory's
method.

[You may not need the polymorphism that the GoF
construction patterns provide, though; you may just
need a "hard-wired" variation on the
concreteFactory object. The GoF patterns are all
geared to dealing with a situation where one needs
to select the right factory based on some dynamic
context. You pretty much know exactly what needs to
be
instantiated as soon as you get a particular Curve
in hand.]

> Yeah, a single CurveBuilder class would probably suffice here, that might have a few functions to build particular types of curves, like spheres. I wonder though, where does reading a curve from a file belong: in the Curve class, or in the CurveBuilder factory?

Factories are ideally suited to instantiation from external configuration data. One gets to hide all the file and parsing stuff away as realized code so that it doesn't distract one from the real problem. The problem solution doesn't care where the objects come from; it just passes messages around among them.

OTOH, if you do use external configuration data, the mapping of identity will probably complicate things just enough so that one would want separate concrete factories. B-) For example, if you have a library class with a getConfigurationLine() method, the parsing of a particular line will be unique to what is being instantiated. Separating and encapsulating those parsing rules is probably a good idea.

> The longer I think about this stuff, the more I see room for improvement. My curves represent dynamic boundaries of a flow domain, and the parameters p and q are more properties of the boundary than of a curve, and also dynamic. So p and q should move into a Boundary class (I hadn't mentioned it in my orignal post, didn't want to make it -too- complicated ;)). I'm still trying to decide whether or not a Boundary Is-A Curve, or Has-A Curve (or Has-A Shape, with Curve a Kind-Of Shape), for instance. Since delegating is usually preferred over inheritance, I am arching towards the latter. In that case, I would like Curves to support external knot vectors too - I need to be able to update the curve shape dynamics quickly, so I don't want unnecessary copying and such. It seems I can move design ideas around like this for ever - I guess it's a sign my design is not yet fully matured ;) It even makes me wonder a bit whether I'm improving my design, or only complicating it... Oh well, all part of the fun I suppose :) Thanks again for your elaborate explanations,

Right; that's Life in Software Development. It never turns out to be as easy as one originally envisioned. Nor is it usually right the first time. B-)

[Maudlin Joycian recollection that dates me... There was a great quote by a fellow names Lahovsky who was one of the designers of the BLISS language. (BLISS was a terrific systems programming language but not very good at anything else.) Lahovsky said, "When I code in BLISS I feel like I am in complete control of the machine. But when I code in Pascal my programs tend to work the first time."]

FWIW, I think you are on the right track with introducing a Boundary critter. When in doubt opt for more objects that are simpler and more cohesive. It is a lot easier to combine the trivial than to the split up

the complex later. I would also prefer delegation wherever reasonable.

****
There is nothing wrong with me that could
not be cured by a capful of Drano.

H. S. Lahman
hsl@xxxxxxxxxxxxxxxxx
Pathfinder Solutions -- Put MDA to Work
http://www.pathfindermda.com
blog: http://pathfinderpeople.blogs.com/hslahman
(888)OOA-PATH

.