

Re: Confusion about splitting classes to allow sharing of resources

Source: <http://coding.derkeiler.com/Archive/General/comp.object/2005-04/msg00526.html>

- *From:* "H. S. Lahman" <h.lahman@xxxxxxxxxxx>
 - *Date:* Fri, 29 Apr 2005 16:12:59 GMT
-

Responding to M.a.stijnman...

This seems very familiar. Didn't I respond to this offline? [I am curious to how my answers line up. B-)]

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.

Re: Confusion about splitting classes to allow sharing of resources

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 defined 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

Re: Confusion about splitting classes to allow sharing of resources

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

Ah, -there- is my inclarity: 'these' refers to the two input vectors, not the interpolation info... The interpolation info depends on the input vectors.

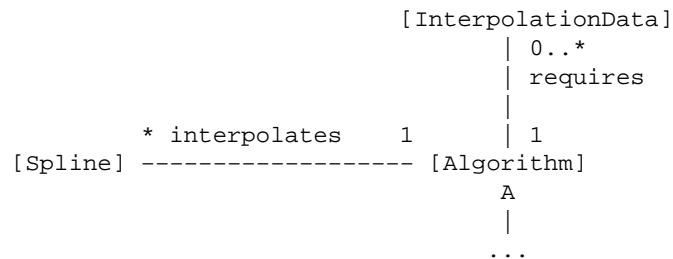
That's fine. They get constructed when one selects the algorithm to use in my model above.

info becomes invalid and will need to be regenerated before any next interpolation can be performed. You are spealing about responsibilities. In the current

Re: Confusion about splitting classes to allow sharing of resources

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:



OK, this was slightly different than my model above. They do the same thing except the model above ties InterpolationData unconditionally to a specific [Algorithm] subclass.

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).

Re: Confusion about splitting classes to allow sharing of resources

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 be regenerated on an as-needed basis.)

It's the responsibility of this 'somebody' I am still a bit confused about - who should this 'somebody' be?

That depends on how one allocates the responsibilities. For example,

case 1: [Spline] controls the overall computation iterations by repeatedly invoking [Algorithm] to do a round of interpolations. In that case [Spline] might poll $t[i]$ and $f[i]$ to see if any changed. It would then invoke `Algorithm.regenerate(<point list>)` prior to the next iteration. [I don't like this because now Spline needs to know about regeneration, which is really an Algorithm issue and a Cubic one at that.]

case 2: [Algorithm] polls $t[i]$ and $f[i]$ for a change flag at the end of each iteration to determine if `Algorithm.regenerate` needs to be called.

case 3: [Cubic] keeps its own list (bitmap) of $t[i]$ and $f[i]$ that it changed and uses that to determine if [InterpolationData] needs to be regenerated.

case 4: in an event-based solution, the $t[i]$ and $f[i]$ elements put a change event on the queue whenever they are modified. When those events are popped for [Algorithm] it does the appropriate regeneration.

Which solution would be best depends on how and when one needs to trigger

Re: Confusion about splitting classes to allow sharing of resources

regeneration of the interpolation values, the basic solution organization (e.g., the nature of the iterations), performance, and all the rest of that good OOA/D/P design stuff. B-)

In the end, though, it is the `t[i]` and `f[i]` elements that know when the values changed. It is the decision to regenerate that may be triggered by any of the objects. One can do that proactively by having the elements themselves announce the change (4), have `[Spline]` trigger it (1), or have `Algorithm` keep track of it somehow (2) and (3).

[Note that if one casts the change behavior into knowledge (e.g., change flags), then one still needs a polling behavior to regenerate and a synchronization behavior to clear those values at the right time (i.e., after all regenerations are completed). Who has responsibility for those activities might be anywhere. But I'll give odds that (2) will be the simplest to implement. B-) One way to view casting the notion of change as knowledge in `t[i]` and `f[i]` is that it allows us to move the processing behavior and decisions to `Algorithm`, who is the one that would logically need have those responsibilities.]

I can define my `Spline` interface to take references/(smart)pointers to the vectors `t[i]` and `f[i]`. This means I don't have to copy data into a spline object. Also, if I have both a function `f(t)` and `g(t)` to interpolate, I can have two splines reference the same `t[i]` vector. All is good. But sooner or later, almost any program will change `t[i]` and/or `f[i]` in some form or fashion, rendering the spline invalid. My question is: who is responsible for keeping track that the spline object is out of date: the spline (possibly by having an observer-like relationship between `t[i]` and `f[i]` vectors and the spline, but that might make operations on the vectors rather inefficient and puts a lot of extra junk into interfaces), or the client?

I don't see any need for copying data unless you need to preserve the original values somehow. (You could even deal with that with a `original_value` and `current_value` attributes.) Let the `Algorithm` navigate to the original vectors through `Spline` and modify them.

Hopefully, I addressed the synchronization mechanism issue directly above. There are three pieces to that issue: knowing what has changed, knowing when to do regeneration, and resetting once regeneration is done. It seems reasonable for `t[i]` and `f[i]` to keep track of whether they have changed. All you need is a boolean `was_changed` attribute to keep track of that. Similarly, it seems intuitive that `Cubic` knows when regeneration might be needed and when it has been completed. That's why I would give odds on (2).

The alternative is to give the spline copies of the data, so no matter what happens to the original data, the spline object still interpolates the old data and thus remains a valid spline at all times. In terms of

Re: Confusion about splitting classes to allow sharing of resources

I had a disconnect here. I thought that the algorithm would be iterative in solving the polynomial so that the $t[i]$ and $f[i]$ data was incrementally modified. However, on each iteration the interpolation data would have to be recomputed for any $t[i]/f[i]$ values that changed in the last iteration.

This seems to be saying that the interpolation data is always regenerated from the original values (old data). If so, I don't understand why it needs to be recomputed at all. Or is there some sort of lag effect going on? (This disconnect no doubt stems from the fact that I don't know enough about spline algorithms to know what the know and function values actually represent.)

In any case, if you need the original data or some lagged values, there are mechanisms for that that can be provided in the $t[i]$ and $f[i]$ elements themselves.

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

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