Subject: Re: Propagating properties
Posted by steinhh on Fri, 16 Oct 1998 07:00:00 GMT

In article <7071ir$or9$1@news.lth.se> Struan Gray
<struan.gray@sljus.lu.se> writes:

```
>    I've decided to set myself a programming excercise in order to
> really get to grips with objects - as opposed to reading the manual
> and fooling myself that I understand it.  I am concentrating on a
> program to draw crystal and surface atomic structures, since the
> structure definitions are nicely hierarchical and can easily be
> objectified, and because I need good 3D plotting so I am forced to use
> object graphics.
```

Ahh - an excellent way to learn objects (actually doing it), and
an interesting application, too. Wish I had the time to do stuff
like this....

```
>     I've been playing with generating arrays of spheres using multiple
> instances of the example 'orb' object.  This works nicely and simply
> but I've hit a conundrum.  I would like to create an ur-orb which
> represents an element, say Si, with properties such as size and colour
> which are adopted by all the instances of Si in the crystal.  A
> problem crops up when after creating the crystal I decide that all the
> Si atoms should be green rather than blue, or that they should be a
> different size.  I would like to be able to just edit the ur-orb and
> have the changes propagate automatically, without having to re-create
> the whole crystal from scratch (which takes some time on my computer).
>
>     I can think of various ways to deal with the problem.  The the two
> most promising are for the ur-orb to maintain a list of dependent
> objects (which seems ugly and slightly defeats the point of object
> programming), or I can subclass 'orb' to create an 'atom' class that
> checks with a specific ur-orb parent every time it draws itself (which
> has a performance hit, and suffers from the lack of a 'copy object' or
> 'copy properties' command discussed here previously).  Does anyone
> have a better idea?
```

Well, somehow, changing the color representation of Si should
be reflected in a change of the color of the polygon(s) that
make up each atom of type Si, before the figure is redrawn.
The question is how to do it.

My $0.05 example:

```
  silicon = obj_new("atom_type",name='Si',color=<yellow>, valence=....)
  lithium = obj_new("atom_type",name='Li',color=<red>, valence=...)
```

```
   atom1 = obj_new("atom",type=silicon,....)
   atom2 = obj_new("atom",type=lithium,....)
```

Each atom keeps a pointer to it's atomic type (there should be just *one* instance of each type!! Look up a thread from some time ago about "singletons" (I think?)).

Now, one of two things should be done:

1. During Atom::Init, a call is made to type->register_me,self
   That is, the "atom_type" object is responsible of keeping
   track of all atoms of its own kind. (Believe me, there's
   nothing non-object oriented about this!)
   When you say e.g.,
   ```
           silicon->setproperty,color=<blue>
   ```
   then silicon immediately tells all the atoms of its own kind
   to make the change in their polygons.

2. The Atom::Draw method needs to be rewritten, to update the
   color of the polygons based on information from a call to
   self.type->getproperty,color=color

Looking at it after writing it down, these two methods are very similar to your own suggestion, just rephrased somewhat... For performance reasons I think I would go for #1. And done this way, it's not at all against the spirit of OO programming. An atomic type, with all of its atomic properties fit together quite naturally - and it's something that you'd like to have more than one instance of, since there are more than one atomic type, so make it an object.

An atom as such is the "prototype" of an object - but it's not necessary to specify e.g. Li as a subclass of "Atom" (though it *could* be done!). However, each atomic instance that represents Li should point to the same "definition object", or else you'd risk having several types of Lithium hanging around!

Regards,

Stein Vidar

Subject: Re: Propagating properties
Posted by Struan Gray on Mon, 19 Oct 1998 07:00:00 GMT
View Forum Message <> Reply to Message

Stein Vidar Hagfors Haugan, steinhh@ulrik.uio.no writes:

> 1. During Atom::Init, a call is made to type->register_me,self
>   That is, the "atom_type" object is responsible of keeping
>   track of all atoms of its own kind. (Believe me, there's
>   nothing non-object oriented about this!)
>   When you say e.g.,
>   silicon->setproperty,color=<blue>
>   then silicon immediately tells all the atoms of its own kind
>   to make the change in their polygons.
>
> 2. The Atom::Draw method needs to be rewritten, to update the
>   color of the polygons based on information from a call to
>   self.type->getproperty,color=color
>
> Looking at it after writing it down, these two methods are very
> similar to your own suggestion, just rephrased somewhat...

I think you said it more clearly :-)

Having RTFM'd over the weekend I have discovered that in this case there is a third possibility. Because all the properties I want to propagate are to do with how the atoms are displayed, I can use the 'atom_type' as a symbol to be plotted at each vertex of a 3D polyline object. This will also make simple bonds easy to draw since a single 'IDLgrPolyline' can contain many individual (and disconnected) line segments.

I don't know yet how much memory and processing overhead this will entail, if any, or how it will affect data-picking once I start allowing the user to interact with the model. I particularly want the user to be able to create things like point defects and dislocations by altering and inserting atoms, and that may be easier to do if I take care of all the objects explicitly myself. I'll play around with the various ideas and see which works best.

Struan